НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

«На правах рукопису»

УДК 519.6

«До захисту допущено»
В.о. завідувача кафедри
_____ М.В.Грайворонський
“___”_____2018 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності: 113 Прикладна математика

на тему: Кластеризація кольорових об'єктів на основі вкладень векторів зображень у нейронних мережах

Виконав (-ла): студент (-ка) 2 курсу, групи ФІ-71мп
(шифр групи)

Чернятевич Антон Андрійович _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник к.ф.-м.н., доц. Орєхов Олександр Арсенійович
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою
Спеціальність (спеціалізація) – 113 Прикладна математика («Математичні методи комп'ютерного моделювання»)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ М.В.Грайворонський
(підпис)

«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Чернятевичу Антону Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації
Кластеризація кольорових об'єктів на основі вкладень векторів зображень у нейронних мережах

науковий керівник дисертації к.ф.-м.н., доц. Орєхов Олександр Арсенійович,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «15 » листопада 2018 р. № 4171-с

2. Термін подання студентом дисертації 10.12.2018 р.

3. Об'єкт дослідження _____
_____

4. Вихідні дані _____
_____
_____
_____

5. Перелік завдань, які потрібно розробити _____
_____
_____
_____
_____
_____

6. Орієнтовний перелік ілюстративного матеріалу _____
_____
_____

7. Орієнтовний перелік публікацій _____

_____

_____

8. Консультанти розділів дисертації*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка

Студент _____ _____
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ _____
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено наукового керівника магістерської дисертації.

РЕФЕРАТ

Кваліфікаційна робота містить: 78 сторінок, 23 рисунки, 3 таблиці, 27 джерел.

Метод вкладень векторів вже досить давно використовується у сфері обробки природної мови для зіставлення слів або окремих словосполучень деякому словнику векторів. Векторні представлення слів і фраз здатні значно поліпшити якість роботи деяких методів автоматичної обробки природної мови. Цей метод знайшов використання і для кольорових зображень, що призвело до активного дослідження функцій втрат нейронних мереж для вирішення задач у сфері машинного зору. Вкладення векторів, отримані з вирішення задачі з учителем із натренованою моделлю на деякій кількості класів з передовими функціями втрат, можуть використовуватися для вирішення задачі кластеризації.

Метою магістерської дисертації є розробка та покращення методу для вирішення наукомісткої задачі аналізу футболістів під час футбольного матчу, а саме кластеризації персон на полі, з характерними вимогами щодо точності та швидкості роботи алгоритму.

У роботі досліджено класичні підходи для вирішення задачі, а також використано сучасні перспективні розробки для тренування та побудови архітектури нейронних мереж.

ВКЛАДЕННЯ ВЕКТОРІВ, МАШИННЕ НАВЧАННЯ, КЛАСТЕРИЗАЦІЯ, НЕЙРОННІ МЕРЕЖІ, РЕАЛЬНИЙ ЧАС

ABSTRACT

The qualifying paper contains 78 pages, 23 figures, 3 tables, 27 sources.

Embeddings method has long been used in the field of natural language processing to match words or individual phrases to a certain vocabulary of vectors. Vector representations of words and phrases can greatly improve the quality of the work of some methods of automatic processing of natural language. This method has also been used for color images, which led to an active study of loss functions in neural networks for solving problems in the field of computer vision. Embeddings that are obtained from solving a supervised learning problem on a certain number of classes with advanced loss functions can be used to solve a clusterization problem.

The purpose of the master's thesis is to develop and improve the method for solving a science-intensive problem of analyzing football players during a football match, namely clustering of persons on the field, with specific requirements regarding the accuracy and speed of the algorithm.

The paper examines the classical approaches to solving the problem, and also uses a modern perspective design for training and building the architecture of neural networks.

CONSTRUCTION OF VECTORS, MACHINE TEACHING, CLUSTERIZATION, NEURAL NETWORKS, REAL TIME

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

грейскейл — зображення у відтінках сірого

датасет — вибірка даних

ембедінг — вкладення векторів

лейбл — розмічена одиниця даних, значення

семпл — вибірка

t-SNE — T-distributed Stochastic Neighbor Embedding

ВСТУП

Актуальність роботи. Метод вкладень векторів вже досить давно використовується у сфері обробки природної мови для зіставлення слів або окремих словосполучень деякому словнику векторів. Векторні представлення слів і фраз здатні значно поліпшити якість роботи деяких методів автоматичної обробки природної мови. Цей метод знайшов використання і для кольорових зображень, що призвело до активного дослідження функцій втрат нейронних мереж останні кілька років.

Суть методу полягає в тому, що на вхід нейронної мережі подається вибірка із кольоровими зображеннями, на яких вона навчається. В процесі навчання ми отримуємо багатовимірний вектор з параметрами мережі — вектор вкладень. Зазвичай архітектура нейронних мереж для задачі класифікації передбачає подальше стискання цього вектора у вектор розмірності кількості лейблів. Така модель буде класифікувати об'єкти лише за тими класами, на яких вона навчалась. Проте якщо не задавати моделі обмеження на кількість класів, тобто не стискати шар з вектором вкладень, це дозволить використовувати модель для вирішення задачі кластеризації на основі інформації параметрів мережі. Тому, в рамках даної роботи, було вирішено розробити метод для автоматичного розбиття довільної кількості об'єктів на класи, а саме поділу футболістів на команди за кольором форми, з використанням вкладень векторів нейронних мереж.

Об'єкт дослідження — різнокольорові зображення довільного розміру

Предмет дослідження — кластеризація кольорових об'єктів на кольорових зображеннях

Мета дослідження. Покращення точності кластеризації за кольором за допомогою використання нейронних мереж та робота алгоритму в режимі реального часу.

Етапи дослідження:

1) ознайомлення зі специфікою задачі кластеризації об'єктів на

зображеннях: аналіз математичних методів кластеризації та методів штучного інтелекту з використанням векторів вкладень нейронних мереж;

2) збір та опрацювання зображень із запису футбольного матчу для навчання моделі нейронної мережі;

3) аналіз відомих архітектур та функцій втрат нейронних мереж, розробка моделі, перевірка її точності на тестувальній вибірці та перевірка швидкості роботи в умовах реального часу;

Наукова новизна одержаних результатів. Розроблений та удосконалений метод вирішення задачі кластеризації персон на полі під час футбольного матчу, яка характеризується вимогами щодо точності та швидкості моделі.

Практичне значення одержаних результатів. Натренована розробленим методом модель використовується для аналізу поведінки футболістів під час матчів компанією SoftConstruct Україна.

1 ОГЛЯД ДЕЯКИХ МЕТОДІВ ТА ПІДХОДІВ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ

1.1 Постановка задачі

Деякі прикладні дослідники, наприклад, як Ендрю Рабинович, вважають, що проблеми у сфері комп'ютерного зору вже вирішені[1] і не потребують такої уваги суспільства, як зараз. Але згідно з ресурсом порівняння напрямів досліджень комп'ютерних наук серед найпопулярніших освітніх закладів[2], сфера машинного зору займає провідні позиції, активно розвивається і набирає ще більшої популярності останніми роками. Проте проблеми, які піднімаються у сучасних наукових працях, та розробка спеціалізованих фреймворків все більше зводяться до вирішення задач більш глибокого розуміння даних, а також розуміння розроблених моделей, їх продуктивності та швидкості роботи. Це зумовлено досить високою якістю розроблених алгоритмів, які вже вирішують більшість проблем машинного зору[3].

У даній роботі основною ціллю є вирішення наукомісткої задачі аналізу поведінки футболістів на футбольному полі під час матчу. Вирішується проблема кластеризації персон на полі, яка також характеризується вимогами щодо високої якості моделі, а також її швидкості роботи.

Вимоги щодо якості досягаються найбільш високим значенням метрики точності моделі, який повинен перевищувати показники класичних підходів щодо вирішення задачі. Швидкість роботи буде вимірюватися згідно зі значенням передачі кадрів у секунду (КВС) відеофіксатором у режимі реального часу, до якого є найменш чутливим людське око, тобто до 25 КВС. Тобто, додаток повинен відпрацьовувати швидше, ніж за 0.04 секунди. Це значення і буде еталонним для вимірювання швидкості моделі.

1.2 Вкладення векторів

Вкладенням векторів (або ембедінгом) у даній роботі називається відображення від дискретних об'єктів, від таких як зображення, до векторів дійсних чисел. Наприклад, 300-мірне вкладення векторів для декількох зображень, який зображено на Рисунку 1.1

	name	values
0	метелик	(0.01359, 0.00075997, 0.24608, ..., -0.2524, 1.0048, 0.06259)
1	автівка	(0.01396, 0.11887, -0.48963, ..., 0.033483, -0.10007, 0.1158)
2	пензлик	(-0.24776, -0.12359, 0.20986, ..., 0.079717, 0.23865, -0.014213)
3	будинок	(-0.35609, 0.21854, 0.080944, ..., -0.35413, 0.38511, -0.070976)

Рисунок 1.1 – Приклад вкладень векторів для 4 зображень у бібліотеці pandas

Індивідуальні розміри у цих векторах, як правило, не мають властивого значення. Натомість це загальні закономірності розташування та відстані між векторами, які використовують у машинному навчанні.[4]

Ембедінги важливі для використання на вхід у систему машинного навчання. Класифікатори та нейронні мережі в загальному випадку, працюють над векторами дійсних чисел. Вони навчаються найкраще за щільних векторів, де всі значення, грубо кажучи, допомагають визначити об'єкт. Проте багато важливих матеріалів для машинного навчання, наприклад, слова тексту, не мають природного векторного уявлення. Функції вкладень векторів є стандартним та ефективним способом перетворення таких дискретних вхідних об'єктів в корисні безперервні вектори.

Вкладення векторів також мають важливе значення як результати машинного навчання. Оскільки ембедінги об'єднують об'єкти у векторі, застосування можуть використовувати подібність у векторному просторі (наприклад, Евклідову відстань або кут між векторами) як надійну та

гнучку міру подібності об'єктів. Одним із поширених завдань є пошук найближчих сусідів.[5] Наприклад, використовуючи ті самі вектори зображень, що описані вище, найближчими сусідами для кожного з них будуть наступні класи та відповідні кути (у градусах):

	name	values
0	метелик	(гусінь, 44.3), (бджола, 50.3), (муха, 53.4)
1	автівка	(джип, 56.6), (мотоцикл, 59.9), (фура, 62.2)
2	пензлик	(олівець, 50.5), (маркер, 52.3), (ручка, 57.0)
3	будинок	(супермаркет, 45.2), (лікарня, 46.8), (цегла, 92.5)

Рисунок 1.2 – Приклад найближчих сусідів для вкладень векторів з Рисунку 1.1 у pandas

Це допоможе застосуванню краще зрозуміти, що автівка та джип в певному сенсі більш схожі (56.6 відстань) від автівки та фури (62.2 відстань).

1.3 Інтерпретація та візуалізація вкладень векторів

На практиці частіше використовуються ембедінги, розмірність яких більше за 3 [6], що є досить великою проблемою для їх розуміння з фізичної точки зору. Тому потрібні методи для їх зчитування, інтерпретації та візуалізації у двох або трьох вимірному просторі.

1.3.1 TensorBoard

Оскільки розробка бібліотек для ефективної роботи з ембедінгами не є ціллю даної роботи, тому доводиться використовувати доступні рішення у відкритому доступі. Одним з таких рішень є TensorBoard, розроблений компанією Google LLC у 2015 році, для їх фреймворку TensorFlow, який

ще буде згадуватися у наступному розділі.

TensorBoard являє собою набір веб-додатків для перевірки та розуміння роботи TensorFlow і графів моделей нейронних мереж. Основний інтерфейс зображено на Рисунку 1.3

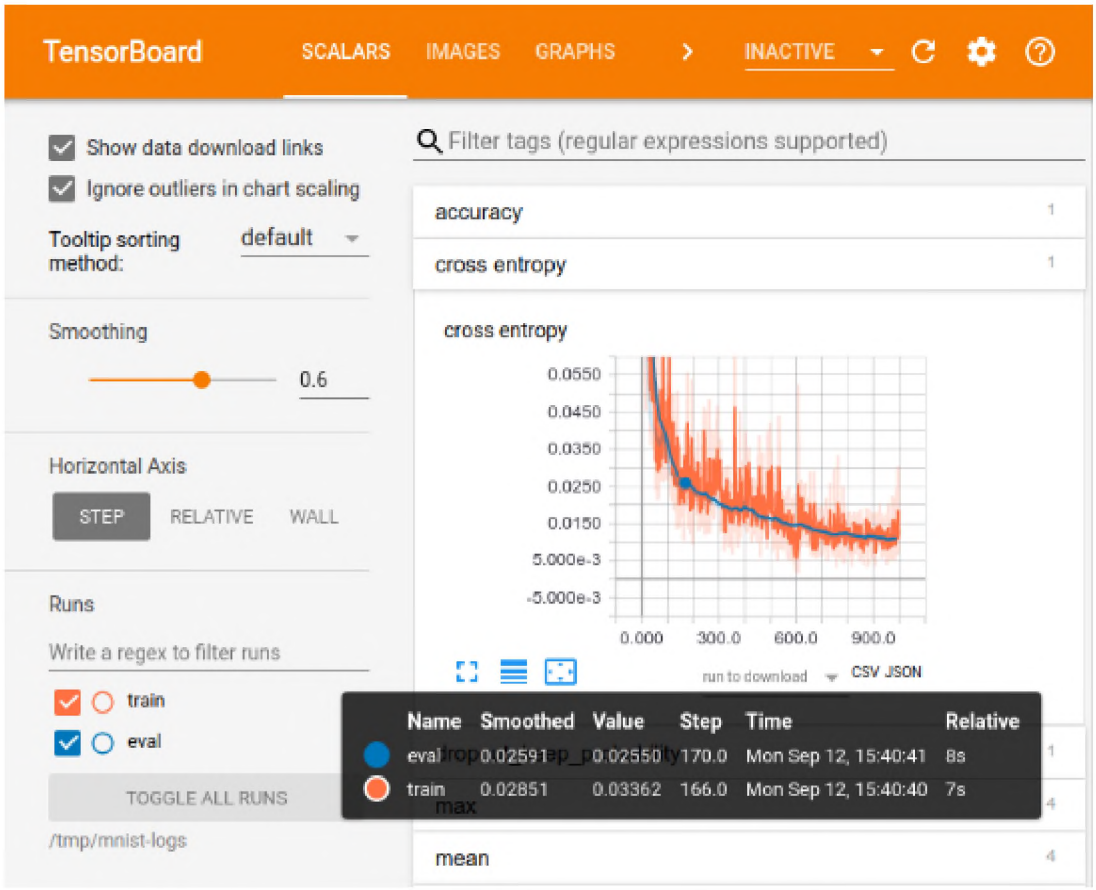

Рисунок 1.3 – Приклад роботи з TensorBoard з офіційної веб-сторінки проекту

Окрім основних переваг: візуалізації графів, підтримки бібліотеки на всіх платформах, дебагінгу, можливості розширення та побудови готового пайплайну, веб-додаток дозволяє ефективно працювати з вкладеннями векторів, а саме: зчитувати, інтерпретувати та візуалізувати їх. Тобто TensorBoard є розв'язком усіх проблем у межах поставленої задачі.

1.3.2 Метод t-SNE

T-SNE — це нелінійний не детерміністичний алгоритм (T-розподілене векторне представлення стохастичного сусіда), який намагається зберегти місцевих сусідів у даних, проте часто шляхом плутанини глобальної структури.[7] Але є можливість обирати, чи потрібно обчислювати дво- або тривимірні проекції. Приклад візуалізації двовимірної проекції зображено на Рисунку 1.4

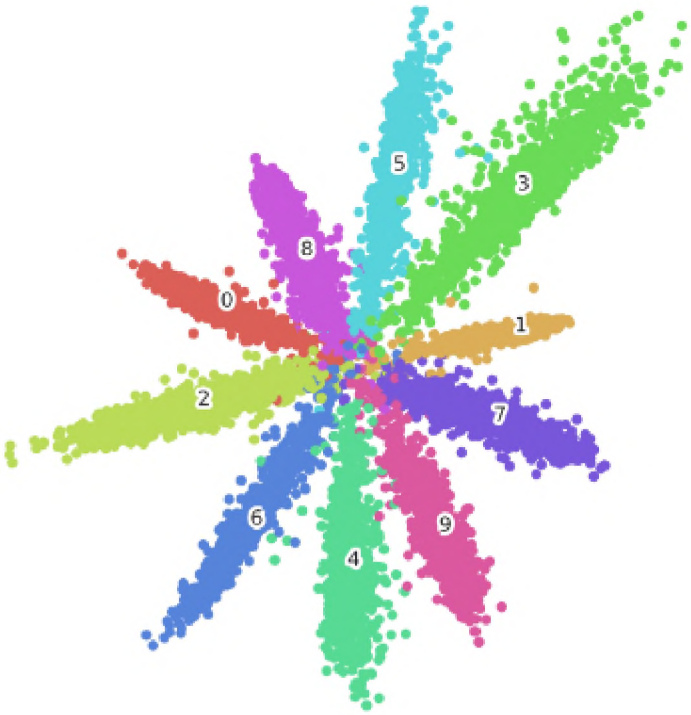

Рисунок 1.4 – Приклад розбиття ембедінгів на датасеті MNIST та їх 2D t-SNE візуалізація

Мета алгоритму полягає в тому, щоб взяти набір точок у багатомірному просторі й знайти вірне представлення цих точок у менш вимірному просторі, як правило, 2D-площині. Алгоритм нелінійний і адаптується до базових даних, виконуючи різні перетворення в різних

регіонах. Ці відмінності можуть бути головним джерелом плутанини.

Особливість t-SNE - це зазвичай змінний параметр "perplexity"(або збентеженість), який відповідає за збалансування уваги між локальними та глобальними аспектами даних. Також, у певному сенсі, є параметром уявлення про кількість близьких сусідів у кожній точці. Значення збентеженості впливає на отримані зображення. У оригінальному документі говориться: "Ефективність SNE досить стійка до змін у незрозумілій ситуації, а типові значення - від 5 до 50."[8] Але історія більш нюансова, ніж це. Найкращий спосіб використання t-SNE може означати аналіз кількох ділянок з різними незручностями.

1.3.3 Метод головних компонент

Метод головних компонент (МГК) — це лінійний детерміністичний алгоритм, який намагається зафіксувати якнайбільше різноманітності даних, наскільки це можливо.[9] МГК, як правило, підкреслює велику структуру даних, але може деформувати локальні залежності. Проектор Вкладень Векторів у TensorBoard вираховує 10 основних компонентів, з яких можна вибрати два або три для перегляду, як зображено на Рисунку 1.5

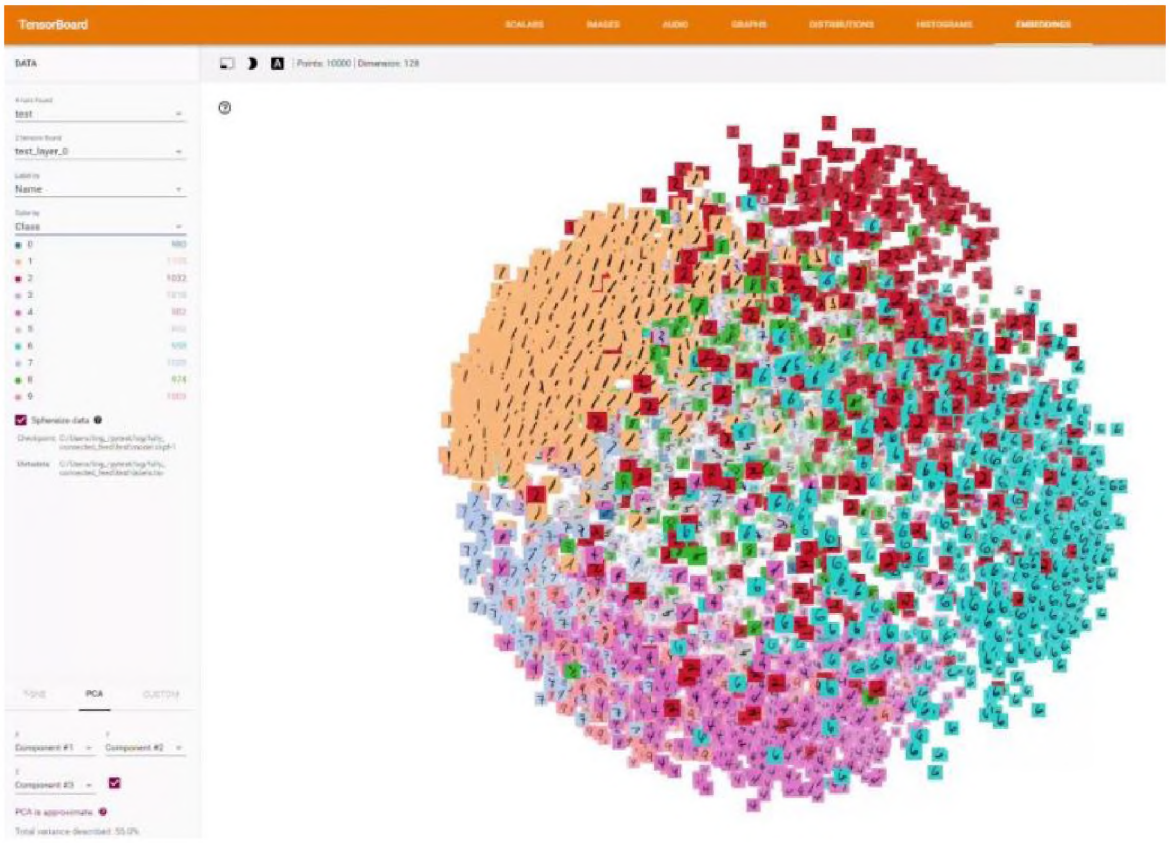

Рисунок 1.5 – Приклад розбиття ембедінгів на датасеті MNIST та їх МГК візуалізація

1.4 Перетворення Фур'є

Перетворення Фур'є перетворює сигнал з часового домену (потужність сигналу як функцію часу) до частотного домену (потужність сигналу як функція частоти). Він показує спектральний вміст сигналу, поділений на дискретні смуги частот (або біни).

Зі сфери обробки звуку перетворення Фур'є має фізично інтуїтивне значення. Звук або функцію $f : [0, b] \to [-1, 1]$ можна представити у вигляді тригонометричного ряду. І кожен термін серії відповідає частоті, яку сприймає людина. Багато ефектів (наприклад, фільтрація, реверберація тощо) мають інтерпретацію в частотній області, яка корисна для аналізу. На жаль, фізичне тлумачення не таке тривіальне, якщо

говорити про зображення. Проте перекладається на аналітичну мову. Якщо ми говоримо про грейскейл, то зображення — це просто функція $f : [0,1]^2 \to [0,1]$. Вона займає точку у квадраті $[0,1] \times [0,1]$ і приймає значення від 0 до 1, що відповідає за інтенсивність. Фур'є-трансформація говорить, що ми можемо представляти цю функцію в частотній області, використовуючи лічильну основу тригонометричних функцій. Наприклад, розмиття зображення відповідає фільтру низьких частот у частотній області.

Проте якщо аналізувати роботу перетворень Фур'є в розрізі кластеризації кольорових зображень, то його використання не задовольнить поставленим вимогам, оскільки будь-які перетворення будуть виконуватися для кожного каналу зображення послідовно. Оскільки таких каналів 3 (розглядається адитивна колірна модель - RGB), а зображень — понад 25 (мінімум 22 футболісти та 3 судді), то на відпрацювання піде набагато більше часу, ніж 0.04 секунди.

1.5 Кольорові гістограми

Коли мова йде про колір, у сфері комп'ютерного зору в першу чергу згадують про кольорові гістограми. Гістограми узагальнюють, скільки разів (частота) кожного значення інтенсивності в зображенні трапляється.[10] Як у розповсюдженому прикладі з будинком, який зображено на Рисунку 1.6, зображення (зліва) має 256 окремих сірих рівнів, а гістограма (справа) показує частоту сірих рівнів.

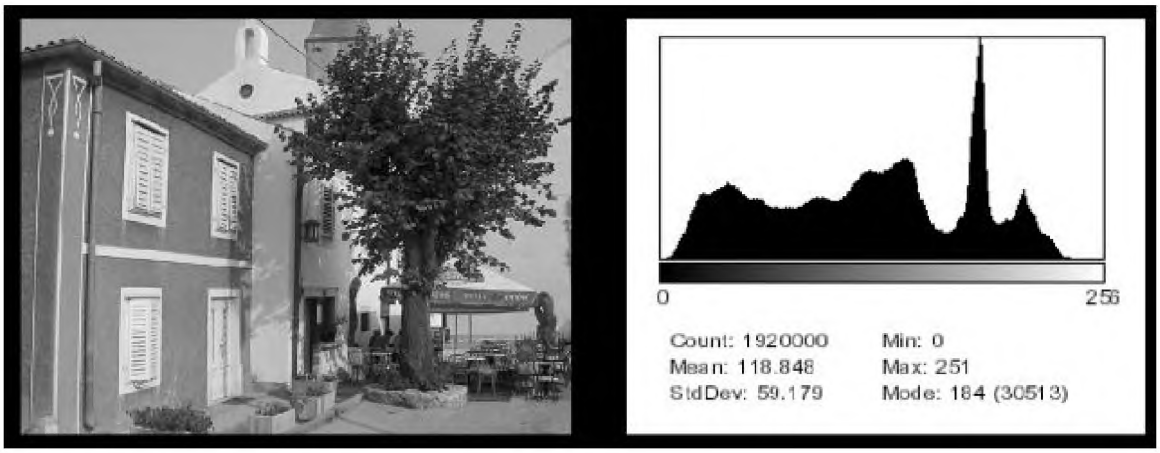

Рисунок 1.6 – Зображення будинку в грейскейлі (зліва) та кольорова гістограма (справа)

Для кольорових зображень адаптивної кольорової моделі RGB таких гістограм може бути всього три, тобто для кожного з каналів.

У реальних умовах, робота з трьома каналами кольорових гістограм не оптимізована у найбільш популярних бібліотеках сфери комп'ютерного зору, які більш детально розглянуті у главі 3.1. Зазвичай є можливість опрацювання лише одного каналу, який, по-перше, не містить всієї інформації про об'єкт на зображенні, що призводить до послідовної обробки всіх каналів, а потім їх комбінування, і, по-друге, є також не оптимізованим для обробки партії зображень.

Для розв'язання першої проблеми було спробувано стиснути кількість рівнів зображення з 256 до 32 методом розбиття їх по 8 різним бінам, а саме зображення перевести з 3 каналів до 1 методом матричних перетворень. Псевдокод цього методу зображено на Алгоритмі 1.1. Але коштом другої проблеми, оптимізація коду мови Python навіть за допомогою мови C не додало швидкості у відпрацюванні всієї програми, оскільки обробка 22 картинок відбувається послідовно. Аналогічні показують себе методи, що вбудовані у фреймворки tensorflow та pytorch. Тобто використання методів для аналізу та обробки гістограм з одним каналом не підходить для вирішення поставленої задачі та потребує пошуку більш потужних бібліотек.

Таблиця 1.1 – Час виконання прорахунку гістограм для одного та партії зображень

Фреймворк/бібліотека	Час виконання (середнє та відхилення за 10000 проходів)
OpenCV (одне зображення)	4.02 мкс +- 99.8 нс
NumPy (одне зображення)	75.3 мкс +- 1.67 мкс
Зменшення каналів гістограми (одне зображення)	9.92 мс +- 89.2 мкс
pyTorch (одне зображення)	9.65 мкс +- 196 нс
ArrayFire (одне зображення)	67.3 мкс +- 1.45 мкс
pyTorch (партія)	4.2 мс +- 62.5 мкс
ArrayFire (партія)	228 мкс +- 5.97 мкс

Algorithm 1.1 Псевдокод на мові Python, функції зменшення кількості бінів у гістограмі, яка залежить від зображення, інтенсивності на 1 бін (тобто 32) та кількості бінів (тобто 8)

```
кількість бінів = np.arange(кількість бінів)
висота, широта, кількість каналів = image.shape
rgb_bin_map = enumerate_bin_maps(кількість бінів, кількість каналів)
зображення = зображення // інтенсивність біна
зображення = [rgb_bin_map.get(x) for x in зображення]
```

Прикладом такої бібліотеки є ArrayFire[11], яка написана на мові C++, але має надбудову на мові Python. У ній реалізовано функціонал обробки партії зображень, але на одно канальних гістограмах також. З останньої, хоч і не досить успішної спроби, було використано частину коду з Алгоритму 1.1 для стиснення трьох вимірної гістограми в одновимірну та подальшого прорахунку на партії зображень. Результатом використання такої бібліотеки є досить непогана швидкість виконання алгоритму, проте не досить висока точність коштом урахування заднього плану картинки. Як зображено на рисунку 1.7, поверхня поля засніжена, а форма футболіста є білою. Тобто, навіть з досить непоганою швидкістю обробки даних, ArrayFire та кольорові гістограми не здатні без додаткових налаштувань знаходити об'єкт кластеризації, а саме футболіста. Але і їх швидкість не є достатньою настільки, щоб виконувати будь-які інші додаткові матричні операції.

Рисунок 1.7 – Приклад зображення поля з засніженою поверхнею

Час виконання гістограм на мові Python за допомогою низки бібліотек та фреймворків зображено у таблиці 1.1. Оскільки деякі методи не мають вбудованого функціонала для обробки партії зображень, то їх не було додано до таблиці, оскільки час виконання рахується за формулою $t_{batch} = t_1 \cdot 300$, де t_1 — час виконання програми на 1 зображенні, а t_{batch} — на партії. Число 30 було обрано довільно, як дуже строгу верхню границю досягнення кількості людей на полі. Цільовий час виконання — приблизно 4 мс, як було описано у розділі 1.1.

1.6 Нейронні мережі

Розглянемо тривіальний[12] приклад нейромережі для вирішення задачі класифікації відповідності футболістів одній команді. Нехай на вхід подається вектор бінарних значень, де перше відповідає за належність футболіста до однієї з команд, друге — за належність іншого, а на виході маємо значення, яке відповідає на питання, чи належать обидва футболісти до однієї команди.

Наприклад, на вході подається вектор $(1, 0)$, який зображено на Рисунку 1.8, а на виході повинні отримати 0. Нехай в мережі тільки один прихований шар, який містить чотири нейрони, а ваги та біас мають випадкові значення.

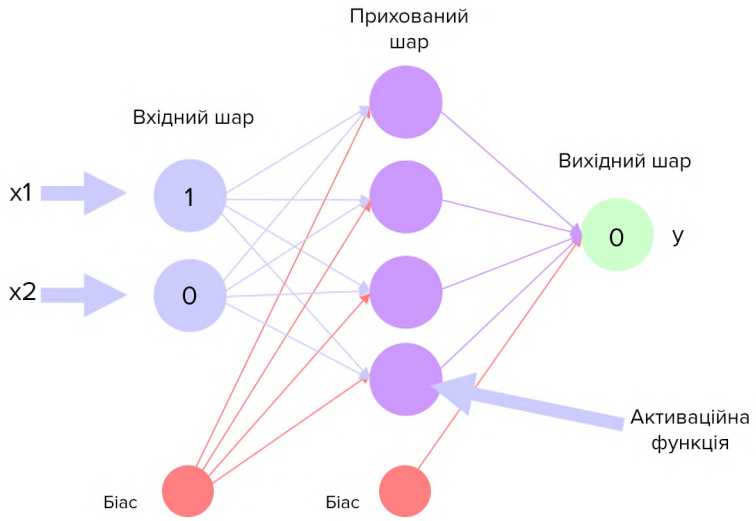

Рисунок 1.8 – Прохід обраного вектора через скритий шар.

1.6.1 Нейронна мережа прямого розповсюдження

У нейронній мережі прямого розповсюдження кожен вихід нейрону стає входом для наступного шару нейронів. У нашому дослідженні вхід до прихованого шару обчислюється за формулою

$$z_{hidden} = \sum_{j=1}^{N} x_j \cdot \omega_j + b_1 \tag{1.1}$$

де N — кількість нейронів у вихідному шарі (дорівнює кількості ознак), x_j — вхідні дані, ω_j — ваги, b_1 — біас.

Нехай ваги та біас мають наступні значення: $\omega_1 = 0.2$, $\omega_2 = 0.4$, $\omega_3 = 0.7$, $\omega_4 = 0.5$, $\omega_5 = 0.3$, $\omega_6 = 0.5$, $\omega_7 = 0.6$, $\omega_8 = 0.9$, $b_1 = 1$. Їх розподіл зображено на Рисунку 1.9.

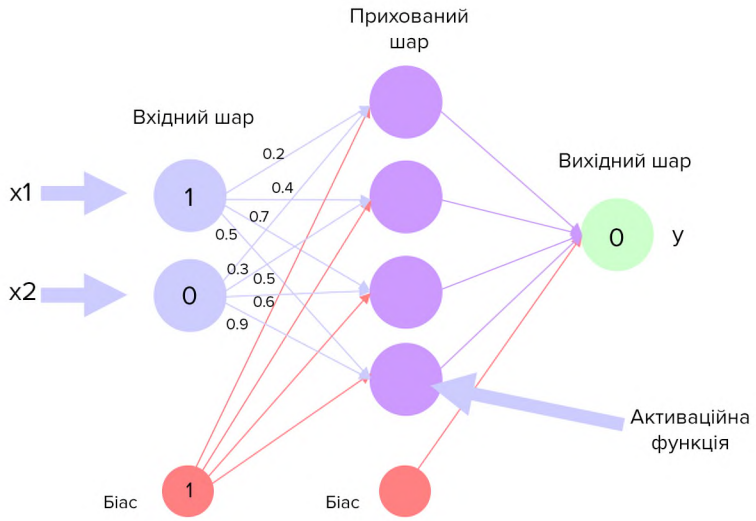

Рисунок 1.9 – Розподіл вагів та біасу під час проходу обраного вектора через скритий шар.

За формулою (1.1) обчислимо вхід до першого нейрону прихованого шару

$$z_{hidden1} = 1 \cdot 0.2 + 0 \cdot 0.3 + 1 = 1.2$$

Аналогічно для інших нейронів:

$$\begin{cases} z_{hidden2} = 1.4 \\ z_{hidden3} = 1.7 \\ z_{hidden4} = 1.5 \end{cases}$$

Кожен нейрон прихованого шару містить активаційну функцію. Вона може бути будь-якою, але зазвичай використовують сигмоід-функцію, яка набуває значень на проміжку $[0; 1]$

$$f(x) = \frac{1}{1 + e^{-x}}$$

Вихід із прихованого шару обчислюється за формулою

$$y_{hidden} = \frac{1}{1 + e^{-z_{hidden}}} \tag{1.2}$$

Застосуємо активаційну функцію (1.2) для першого нейрону прихованого шару

$$y_{hidden1} = \frac{1}{1 + e^{-1.2}} = 0.769$$

Аналогічно для інших нейронів:

$$\begin{cases} y_{hidden2} = 0.802 \\ y_{hidden3} = 0.846 \\ y_{hidden4} = 0.818 \end{cases}$$

Отже, маємо всі виходи нейронів прихованого шару. Тепер залишається тільки обчислити загальний вихід нейронної мережі.

Вхід до вихідного шару обчислюється за формулою

$$z_{out} = \sum_{j=1}^{K} y_{hidden_j} \cdot \omega_j + b_2 \tag{1.3}$$

де K — кількість нейронів у прихованому шарі, $j = \overline{1,4}$

Вихід нейромережі обчислюється за формулою

$$y_{out} = \frac{1}{1 + e^{-z_{out}}} \tag{1.4}$$

Нехай нові ваги та біас в прихованому шарі мають значення: $\omega_1 = 0.2$, $\omega_2 = 0.4$, $\omega_3 = 0.6$, $\omega_4 = 0.8$, $b_2 = 1$. Їх розподіл зображено на Рисунку 1.10.

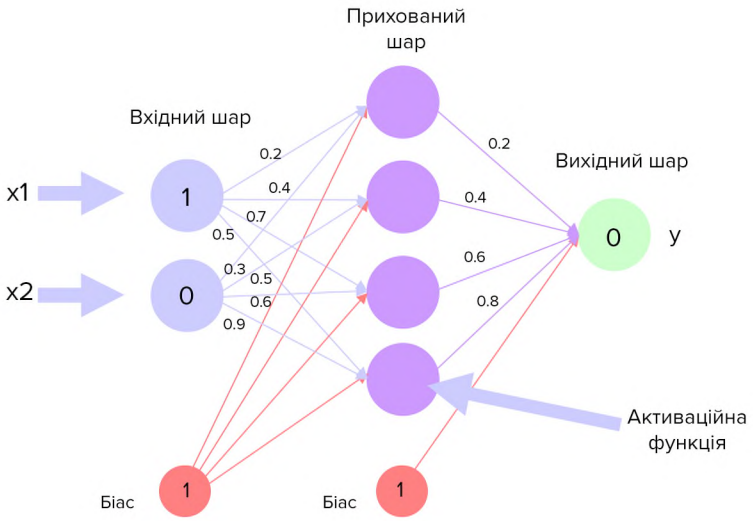

Рисунок 1.10 – Розподіл вагів та біасу під час проходу до вихідного шару.

За формулами (1.3) та (1.4) порахуємо вихід нейронної мережі y_{out}

$$z_{out} = 0.769 \cdot 0.2 + 0.802 \cdot 0.4 + 0.846 \cdot 0.6 + 0.818 \cdot 0.8 + 1 = 2.637$$

$$y_{out} = 0.933$$

Таким чином, отримано значення виходу, яке відрізняється від очікуваного 0. Наступним кроком рахується помилка між очікуваним значенням виходу і отриманим, використовуючи квадратичну функцію помилки

$$E = \frac{1}{2}(y_{expected} - y_{out})^2 \tag{1.5}$$

1.6.2 Зворотнє поширення помилки

У нашій модифікації, нам потрібно, щоб помилка була мінімальною для кожного нейрону і нейромережі в цілому. Для цього треба скорегувати роботу нейромережі, тобто знайти оптимальні ваги та біас, за яких функція помилки набуває мінімального значення. Застосуємо метод градієнтного спуску.

Щоб дізнатись який вклад в помилку робить та чи інша вага, потрібно обчислити часткову похідну від функції помилки за цією вагою.

1) Для вагів вихідного шару

$$\frac{\partial E}{\partial \omega_k} = \frac{\partial E}{\partial y_{out}} \cdot \frac{\partial y_{out}}{\partial z_{out}} \cdot \frac{\partial z_{out}}{\partial \omega_k} \tag{1.6}$$

де $k = 9, 10, 11, 12$

$$\frac{\partial E}{\partial y_{out}} = -(y_{expected} - y_{out})$$

$$\frac{\partial y_{out}}{\partial z_{out}} = y_{out} \cdot (1 - y_{out})$$

$$\frac{\partial z_{out}}{\partial \omega_k} = y_{hidden_j}, j = \overline{1, K}$$

Тобто формула (1.6) має вигляд

$$\frac{\partial E}{\partial \omega_k} = -y_{hidden_j} \cdot y_{out}(1 - y_{out})(y_{expected} - y_{out}) \tag{1.7}$$

Нові ваги розраховуються за формулою

$$\widehat{\omega_k} = \omega_k - \eta \cdot \frac{\partial E}{\partial \omega_k} \tag{1.8}$$

де η — швидкість навчання (зазвичай < 1)

Для розрахунку нового біасу $\widehat{b_2}$ використовуються формули (1.6) - (1.8), але замість ω_k підставляється b_2.

2) Для вагів прихованого шару

$$\frac{\partial E}{\partial \omega_k} = \frac{\partial E}{\partial y_{hidden_j}} \cdot \frac{\partial y_{hidden_j}}{\partial z_{hidden_j}} \cdot \frac{\partial z_{hidden_j}}{\partial \omega_k} \tag{1.9}$$

де $k = \overline{1,8}$, $j = \overline{1,K}$

$$\frac{\partial E}{\partial y_{hidden_j}} = \frac{\partial E}{\partial y_{out}} \cdot \frac{\partial y_{out}}{\partial z_{out}} \cdot \frac{\partial z_{out}}{\partial y_{hidden_j}} = -\omega_{hidden_j} \cdot y_{out}(1 - y_{out})(y_{expected} - y_{out})$$

$$\frac{\partial y_{hidden_j}}{\partial z_{hidden_j}} = y_{hidden_j}(1 - y_{hidden_j})$$

$$\frac{\partial z_{hidden_j}}{\partial \omega_k} = x_k$$

Тобто формула (1.9) має вигляд

$$\frac{\partial E}{\partial \omega_k} = -x_k \cdot \omega_{hidden_j} \cdot y_{hidden_j}(1 - y_{hidden_j}) \cdot y_{out}(1 - y_{out})(y_{expected} - y_{out}) \tag{1.10}$$

Нові ваги розраховуються за формулою (1.8). Новий біас $\widehat{b_1}$ розраховується аналогічно $\widehat{b_2}$. Тобто, перераховується за формулами (1.9) та (1.10) та є різним для кожного нейрону прихованого шару.

Таким чином, ми отримали нейронну мережу із новими, перерахованими параметрами. Подібні нейронні мережі ще називають Vanilla Neural Networks, тобто базовими. Більш досконалими та складними класами нейронних мереж є рекурентні та конволюційні нейронні мережі.

1.7 Рекурентні нейронні мережі

Відповідно до Karpathy[13], насамперед, RNN — це послідовності. Разливе обмеження Vanilla Neural Networks в можливостях їх API: вони

приймають вектор фіксованого розміру як вхідні дані (наприклад, зображення) і створюють вектор фіксованого розміру як результат (наприклад, ймовірності різних класів). Мало того, ці моделі виконують відображення з використанням фіксованої кількості обчислювальних кроків (наприклад, кількості шарів в моделі). Основною особливістю RNN є те, що вони дозволяють нам працювати з послідовностями векторів: послідовності на вході, на виході або в найзагальнішому випадку - з обох кінців.

На Рисунку 1.11 розглянуто декілька прикладів і важливим моментом є те, що на жодній з архітектур не були заздалегідь задані обмеження для довжини послідовностей, оскільки рекурентне перетворення (зелений колір) є фіксованим і може застосовуватися нескінченну кількість раз.

Основою RNN є помилково простий API: вони приймають вхідний вектор X і повертають вихідний вектор y. Однак важливо, що на вміст цього вихідного вектора впливають не тільки дані, які щойно потрапили на вхід, але також і всі попередні входи, тобто які ми подавали в минулому. Розглянемо на прикладі одноступеневої функції прихованого шару для нейромережі прямого розповсюдження. На вхід подається вектор x, прихований стан змінюється за формулою

$$h_t = tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t) \tag{1.11}$$

та y на виході обчислюється множенням матриці на вектор стовпчик

$$y = W_{hy} \cdot h_t \tag{1.12}$$

де параметрами RNN є три матриці W_{hh}, W_{xh}, W_{hy}. Функція $tanh$ додає нелінійної природи та стискає активаційні значення на проміжку $[-1; 1]$. Тобто вихід залежить від попереднього прихованого стану та поточного вводу, які взаємодіють між собою та стискаються $tanh$ у новий вектор станів.

Ми задаємо матриці RNN з випадковими числами і велика частина

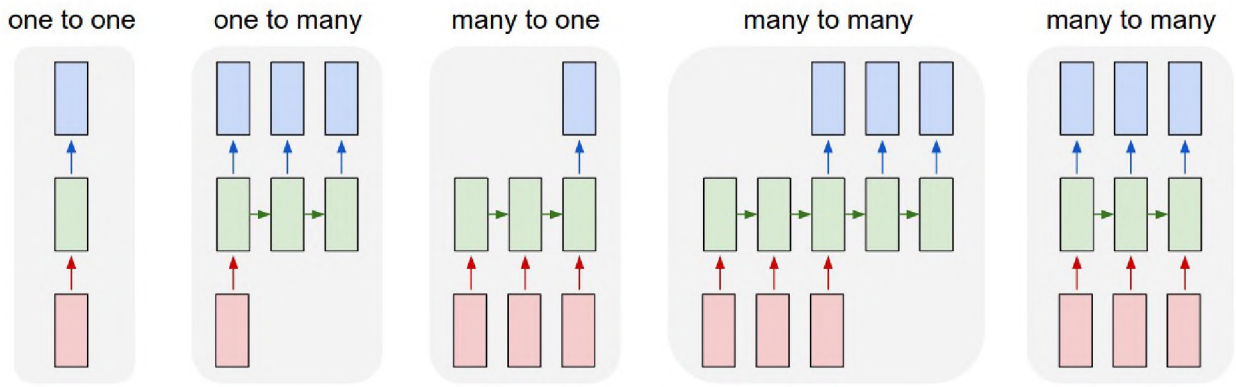

Рисунок 1.11 – Кожен прямокутник є вектором, а стрілки представляють функції (наприклад, множення матриць). Вхідні вектори виділені червоним кольором (блоки знизу), вихідні вектори - синім (блоки згори), і зеленим (блоки посередині)- стан векторів RNN. Зліва направо: (1) Vanilla режим обробки без RNN, від введення фіксованого розміру до виведення фіксованого розміру (наприклад, класифікація зображень). (2) Послідовності на виході (наприклад, розпізнавач картинки приймає зображення і виводить речення слів). (3) Послідовності на вході (наприклад, аналіз настроїв, коли задана послідовність класифікується як вираз позитивного або негативного почуття). (4) Послідовності на вході та на виході (наприклад, машинний переклад: RNN зчитує речення англійською мовою і потім виводить його французькою мовою). (5) Вхід і вихід з синхронною послідовністю (наприклад, класифікація відео, де ми хочемо позначити кожен кадр).

роботи під час навчання витрачається на пошук саме тих матриць, які призводять до бажаних результатів. Якість результату вимірюється за допомогою деякої функції втрат, яка наглядно демонструє, які саме виходи y хотілось би бачити у відповідь на вхідні послідовності x.

В решті решт, RNN - це нейронні мережі і все працює монотонно краще (за умови, що все зроблено вірно), якщо поєднувати разом приховані шари. Наприклад, сформувати двошарову нейронну мережу таким чином, що

вихід (1.12), який позначимо y_1, буде новим входом замість x, тобто

$$h_t = tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot y_{1_t})$$

$$y = W_{hy} \cdot h_t$$

Згідно з Karpathy[13] на практиці частіше використовують інший тип рекурентних мереж — LSTM, які дають кращі результати внаслідок більш досконалого рівняння оновлення та більшої динаміки у методі зворотнього поширення помилки. Тобто основними відмінностями є більш складна форма h_t у виразі (1.11) та можливості роботи пам'яттю

1) обирати те, що запам'ятовує нейромережа;

2) обирати те, що забуває нейромережа;

3) обирати в якому об'ємі пам'ять повинна подаватись на вихід.

Проте використання LSTM нейронних мереж у додатках, які працюють у режимі реального часу, — є досить спірним питанням. Через специфічну архітектуру та зазвичай великі розміри вкладень векторів, кластеризація об'єктів займає більше часу, ніж швидкість передачі кадрів у вхідному відео. Навіть достатньо малий розмір зображень (40x25 по висоті та ширині відповідно) у партії (22 зображення, оскільки 22 футболісти) на вході у мережу повинні оброблятися менше, ніж за 0.04 секунди. Тобто класичні підходи до побудови архітектури LSTM нейронної мережі не підходять в рамках даної задачі, а стиснення та оптимізація такої архітектури не є об'єктом дослідження даної роботи.

Але високої точності та швидкості кластеризації можна досягти за допомогою більш прямого підходу, а саме кластеризації на кожному кадрі, або партії кадрів, які приходять на вхід нейронної мережі під час передачі відео потоку у додаток. У такому випадку можна обрати одну з невеликих архітектур конволюційних нейронних мереж вихідний код яких є у відкритому доступі.

1.8 Конволюційні нейронні нережі

Конволюційні нейронні мережі (КНН) складаються з одного або декількох згорткових шарів спочатку, а потім один або декілька повністю з'єднаних шарів, як у стандартній багатошаровій нейронній мережі. Архітектура КНН призначена для використання двовимірної структури вхідного зображення (або іншого двовимірного входу, такого як мовний сигнал, наприклад). Це досягається за допомогою локальних зв'язків та зв'язних ваг, за якими йде певна форма об'єднання, що призводить до виникнення інваріантних особливостей (або ознак). Ще однією перевагою КНН є те, що їх простіше тренувати та вони мають набагато менше параметрів, ніж повністю зв'язані мережі з однаковою кількістю прихованих блоків. [14]

Вхід до конволюційного шару - це зображення $h \cdot w \cdot c$, де h – висота зображення, w – ширина, а c – кількість каналів. Наприклад, RGB-зображення має $c = 3$. Конволюційний шар має k фільтрів (або ядер) розміром $m \cdot n \cdot q$, де m та n менше, ніж розмір зображення, а q може бути таким самим, як кількість каналів c або менше, і може змінюватися для кожного ядра. Розмір фільтрів дає початкову локально зв'язану структуру, кожен з яких згортається з зображенням, для створення k мап ознак розміром $h - m + 1$ або $w - n + 1$. Кожна така мапа потім підсемплюється зазвичай з середнім або максимальним об'єднанням (або пулінгом) по $p \cdot p$ сусідніх областей, де p змінюється від 2 для малих зображень (наприклад, датасет MNIST, про який піде мова у наступних розділах) і зазвичай не більше 5 для більших входів. Як до, так і після шару підсемплінгу для кожної мапи об'єкта застосовується біас та сигмоідозалежна нелінійність. На рисунку 1.12 показано повний шар КНН, який складається зі згорткових та підсемплових підшарів.

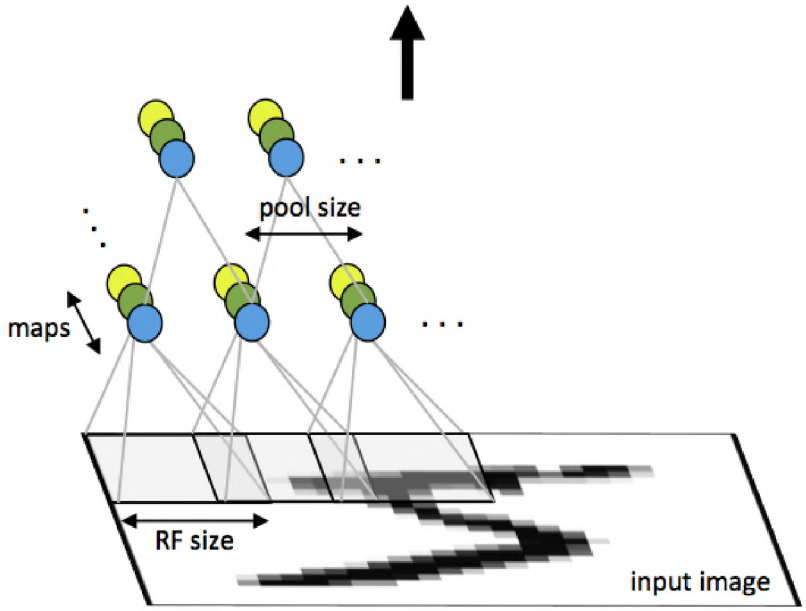

Рисунок 1.12 – Перший шар згорткової нейронної мережі з пулінгом. Одиниці одного кольору мають пов'язані ваги, а одиниці різного кольору – різні мапи фільтрів.

Таким чином, останній повністю зв'язний шар дозволяє корегувати розмір нейронної мережі за допомогою кількості фільтрів, тобто надає можливості змінювати розмірність шару вкладень векторів. Для задачі з великою кількістю класів її слід збільшувати, тобто робити вихідне значення фільтрів рівним, наприклад, 1024. Проте для випадку даної задачі та зібраного датасету, про який піде мова у розділі 3, вистачає і 64 фільтрів. Останнє значення отримано емпіричним шляхом.

Висновки до розділу 1

У даному розділі була сформульована задача даної роботи, а також було розглянуто перетворення Фур'є, кольорові гістограми та сучасні методи машинного навчання з ідеєю кластеризації за допомогою їх векторів вкладень. Також було докладно розібрано модель одношарової нейронної мережі прямого розповсюдження зі зворотним поширенням

помилки, приведені переваги рекурентних та конволюційних нейронних мереж.

Задача кластеризації за кольором була сформульована вже досить давно[15], проте розвиток технологій[16] потребує все більш досконалих розв'язків цього питання. Технологічні досягнення вимагають не лише якості моделей, але ще і компактності та швидкої роботи. Це робить машинне навчання фаворитом над іншими методами та підходами через різноманітність та оптимізацію своїх алгоритмів і фреймворків. А використання векторів вкладень нейронних мереж дозволяє вирішити дану задачу явним методом. Тобто, навчити модель звертати увагу саме на об'єкти кластеризації — футболістів — без урахування інших більш волатильних об'єктів, таких як, наприклад, реклама на полі або патерни поля. Унікальність цього підходу робить вкладення векторів більш універсальним концептом, ніж просто шаром у черговій архітектурі нейронної мережі.

2 ФУНКЦІЇ ВТРАТ

2.1 Функція втрат у нейронних мережах

Функція втрат є важливою частиною в штучних нейронних мережах, яка використовується для вимірювання невідповідності між прогнозованим значенням $(\hat{y})$ та фактичним лейблом (y)[17]. Це невід'ємне значення, де робастність моделі збільшується разом зі зменшенням значення функції втрат. Функція втрат є ядром функції емпіричного ризику, а також важливою складовою функціонування структурного ризику. Як правило, функція структурного ризику моделі складається з емпіричного терміну ризику та терміну регуляризації, який можна представити як

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta) + \lambda \cdot \Phi(\theta) \tag{2.1}$$

$$= \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L\big(y^{(i)}, \hat{y}^{(i)}\big) + \lambda \cdot \Phi(\theta) \tag{2.2}$$

$$= \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L\big(y^{(i)}, f(\mathbf{x}^{(i)}, \theta)\big) + \lambda \cdot \Phi(\theta) \tag{2.3}$$

де $\Phi(\theta)$ — це термін регуляризації, θ — це параметри моделі, яку потрібно навчити, $f(\cdot)$ — функція активації та $\mathbf{x}^{(i)} = \{x_1^{(i)}, x_2^{(i)}, \ldots, x_m^{(i)}\} \in \mathbb{R}^m$ — тренувальна вибірка.

Проте подальший опис зосереджуватиметься лише на емпіричному терміні ризику (функція втрат) $\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} L\big(y^{(i)}, f(\mathbf{x}^{(i)}, \theta)\big)$, а також математичних виразах кількох часто використовуваних функцій втрат.

2.1.1 Середня квадратична похибка

Середня квадратична помилка (СКП) або квадратична функція втрат широко використовується в лінійній регресії, як показник продуктивності, а метод мінімізації СКП називається звичайними найменшими квадратами (ЗНК)[18]. Основний принцип ЗНК полягає в тому, що оптимізована фітингова лінія повинна мінімізувати суму відстані кожної точки до лінії регресії, тобто мінімізувати квадратичну суму. Стандартна форма функції втрат СКП визначається як

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

де $(y^{(i)} - \hat{y}^{(i)})$ називається залишком, а ціллю функції втрати СКП є мінімізація залишкової суми квадратів. Однак, якщо використовувати сигмоїд як функцію активації, СКП буде афектована проблемою повільної конвергенції (тобто швидкості навчання). Для інших функцій активації вона не матиме такої проблеми.

Наприклад, за допомогою сигмоїда $\hat{y}^{(i)} = \sigma(\mathbf{z}^{(i)}) = \sigma(\theta^T\mathbf{x}^{(i)})$, якщо розглядати лише один зразок, скажімо, $(y - \sigma(\mathbf{z}))^2$, та його похідна обчислюється за

$$\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\theta}} = -(y - \sigma(\mathbf{z})) \cdot \sigma'(\mathbf{z}) \cdot \mathbf{x}$$

відповідно до форми та особливості сигмоїда, коли $\sigma(\mathbf{z})$ має тенденцію приймати значення 0 або 1, $\sigma'(\mathbf{z})$ близький до нуля, а коли $\sigma(\mathbf{z})$ близько 0.5, $\sigma'(\mathbf{z})$ досягає цього максимуму. У такому випадку, коли різниця між прогнозованим значенням і справжнім лейблом $(y - \sigma(\mathbf{z}))$ велика, $\sigma'(\mathbf{z})$ буде близьким до 0, що зменшує швидкість конвергенції, а це неправильно, оскільки очікується, що швидкість навчання повинна бути швидкою, коли велика помилка.

2.1.2 Крос-ентропія

Згідно з MIT Press[19], крос-ентропія зазвичай використовується в бінарній класифікації (припускається, що лейбли приймають значення 0 або 1) як функція втрат (для багатоканальної класифікації використовується Багато-класова крос-ентропія), яка обчислюється наступним чином

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$$

Крос-ентропія вимірює розбіжність між двома розподілами ймовірності. Якщо величина крос-ентропії є великою — це означає, що різниця між двома розподілами велика, тоді як, якщо крос-ентропія мала — це означає, що два розподіли схожі один на одного. Як вже згадувалось у СКП, якщо використовується сигмоїд як функція активації, це призводить до повільної дивергенції, проте крос-ентропія не має такої проблеми. Так само, якщо ми маємо $\hat{y}^{(i)} = \sigma(\mathbf{z}^{(i)}) = \sigma(\theta^T\mathbf{x}^{(i)})$ і розглядаємо лише одну навчальну вибірку, використовуючи сигмоїд, у нас є $\mathcal{L} = y\log(\sigma(\mathbf{z})) + (1-y)\log(1-\sigma(\mathbf{z}))$, то похідна обчислюється як

$$\frac{\partial\mathcal{L}}{\partial\theta} = (y-\sigma(\mathbf{z}))\cdot\mathbf{x}$$

Порівнюючи з похідною в СКП, це виключає термін $\sigma'(\mathbf{z})$, де швидкість навчання контролюється лише з $(y-\sigma(\mathbf{z}))$. У цьому випадку, коли різниця між прогнозованим та фактичним значеннями є великою, швидкість навчання, тобто швидкість конвергенції, є великою. В іншому випадку, коли різниця невелика, швидкість навчання є малою, що і є нашим сподіванням. Взагалі, у порівнянні з функцією квадратичних втрат, функція крос-ентропії має переваги, що забезпечують швидку конвергенцію та досягають глобальної оптимізації (наприклад, імпульс, це збільшує крок оновлень).

2.1.3 Косинус близькості

Функція втрат косинусу близькості вираховує косинусну близькість між прогнозованим значенням і фактичним значенням, яка визначається як

$$\mathcal{L} = -\frac{\mathbf{y} \cdot \hat{\mathbf{y}}}{||\mathbf{y}||_2 \cdot ||\hat{\mathbf{y}}||_2} = -\frac{\sum_{i=1}^{n} y^{(i)} \cdot \hat{y}^{(i)}}{\sqrt{\sum_{i=1}^{n} \left(y^{(i)}\right)^2} \cdot \sqrt{\sum_{i=1}^{n} \left(\hat{y}^{(i)}\right)^2}}$$

де $\mathbf{y} = \{y^{(1)}, y^{(2)}, \ldots, y^{(n)}\} \in \mathbb{R}^n$. Це те саме, що і функція втрат косинусу подібності[20], яка є мірою подібності між двома ненульовими векторами предгільбертового простору, що вимірює косинус кута між ними. Векторні одиниці максимально "подібні", якщо вони паралельні, та максимально "різні", якщо вони ортогональні. А це аналогічно косинусу, який є одиницею, коли нульовий кут між відрізками, і є нульовим, коли відрізки перпендикулярні.

2.1.4 Триплет

Тренування нейронної мережі із функцією втрат триплет відрізняється від інших. Функція приймає зображення на якір та порівнює його як з позитивним зразком, так і з негативним зразком. Але різниця між зображенням прив'язки та зображенням з таким самим лейблом має бути низькою, проте як різниця між зображенням прив'язки та негативним — високою. Таким чином триплет мінімізує відстань між об'єктами одного класу, та максимізує між об'єктами різних. Функція Триплет задається наступним чином:

$$\mathcal{L} = \max(d(a, p) - d(a, n) + margin, 0)$$

де a – це якір, p – позитивне зображення, тобто схоже до якоря, та n – негативне зображення. Відомо, що невідповідність між a і p повинна

бути меншою, ніж різниця між a і n. Інша змінна, $margin$, яка є гіперпараметром, додається до функції втрат і визначає, наскільки далеко повинні бути відмінності. Наприклад, якщо $margin = 0.2$ і $d(a, p) = 0.5$, то $d(a, n)$ повинна бути, щонайменше, рівна 0.7. Тобто $margin$ допомагає нам розрізняти два зображення краще.

Тому, використовуючи цю функцію втрат, ми обчислюємо градієнти і за допомогою градієнтів ми оновлюємо ваги та біас мережі. Для навчання мережі ми беремо якірне зображення та випадково набираємо позитивні та негативні зображення, обчислюємо функцію втрат і оновлюємо його градієнти. Візуалізація методу зображена на рисунку 2.1.

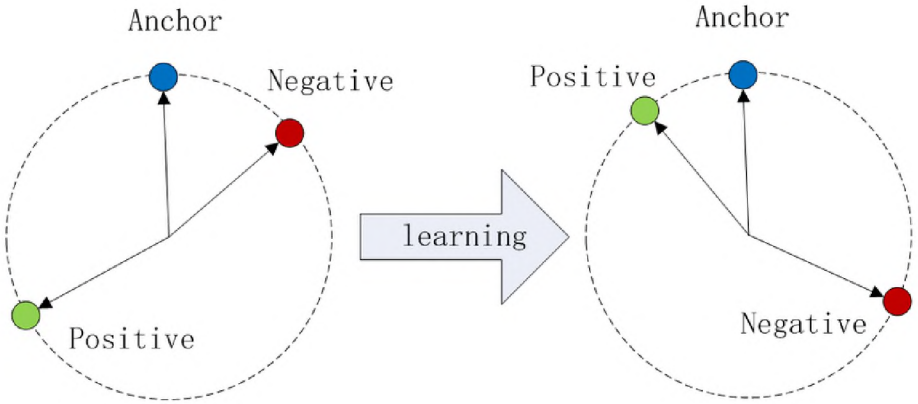

Рисунок 2.1 – Принцип роботи тріплет функції втрат

Але ця функція втрат використовується у так званих Сіамських нейронних мережах. Але їх специфіка не враховується у даному досліджені, тому функція втрат триплет використовується під час валідації результатів тренування нейронної мережі, але не як класична функція втрат. Такий метод валідації разом з контекстним вдосконаленням результатів, яке детальніше описується у розділі 3, призводить до покращення загальної точності розробленої моделі.

2.2 Функція втрат LSoftmax

У 2017 році Лью та ін.[21] запропонував модифікацію функції стиснення розмірності Софтмакс, яку використовував як функцію втрат. Повна назва цієї функції — Large Margin Softmax Loss, скорочено — LSoftmax. Ця функція покращує генералізованість конволюційних нейронних мереж. Приклад покращення на датасеті MNIST зображено на рисунку 2.2.

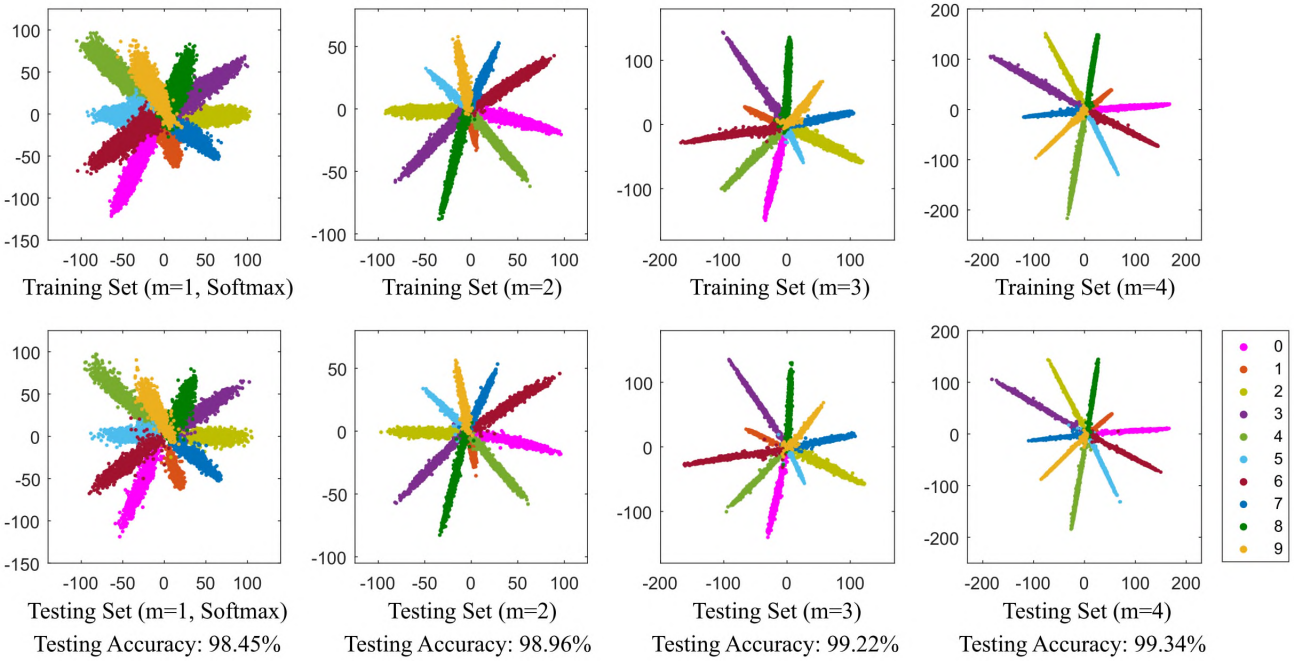

Рисунок 2.2 – Приклад роботи функції втрат LSoftmax з оригінального репозиторію[22] по результатам самого Лью

Але перед відтворенням програмної реалізації цього метода, необхідно розібрати формули, вказані у статті[21], та максимально їх спростити.

Задаємо функцію:

$$w^T x = |w||x|cos\theta$$

Тоді знайдемо похідну функцію втрат J в залежності від x. Вводимо

$$\frac{\partial |x|}{\partial x} = \frac{x}{|x|}$$

Тоді можемо спростити наступні вирази:

$$\frac{\partial cos\theta}{\partial x} = \frac{\partial}{\partial x}\left(\frac{w^T x}{|w||x|}\right) = \frac{w}{|w||x|} - \frac{(w^T x)x}{|w||x|^3} \tag{2.4}$$

$$\frac{\partial sin^2\theta}{\partial x} = \frac{\partial}{\partial x}\left(1 - cos^2\theta\right) = -2cos\theta\frac{\partial cos\theta}{\partial x} \tag{2.5}$$

$$cos(m\theta) = \sum_{n=0}^{\lfloor \frac{m}{2} \rfloor}(-1)^n \binom{m}{2n}(cos\theta)^{m-2n}(sin^2\theta)^n \tag{2.6}$$

$$\frac{\partial cos(m\theta)}{\partial x} = m(cos\theta)^{m-1}\frac{\partial cos\theta}{\partial x} + \tag{2.7}$$

$$+ \sum_{n=1}^{\lfloor \frac{m}{2} \rfloor}(-1)^n \binom{m}{2n} \cdot [n(cos\theta)^{m-2n}(sin^2\theta)^{n-1}\frac{\partial sin^2\theta}{\partial x} + \tag{2.8}$$

$$+ (m-2n)(cos\theta)^{m-2n-1}(sin^2\theta)^n\frac{\partial cos\theta}{\partial x}] \tag{2.9}$$

Підставивши все в функцію (11) з [21], отримуємо наступне:

$$f = (-1)^k|w||x|cos(m\theta) - 2k|w||x| = \left[(-1)^k cos(m\theta) - 2k\right]|w||x|$$

$$\frac{\partial f}{\partial x} = \left[(-1)^k cos(m\theta) - 2k\right]\frac{|w|}{|x|}x + (-1)^k|w||x|\frac{\partial cos(m\theta)}{\partial x}$$

Таким чином отримуємо похідну для функції втрат J по x:

$$\frac{\partial J}{\partial x_i} = \sum_{j, j \neq y_i} \frac{\partial J}{\partial f_{i,j}} \cdot \frac{\partial f_{i,j}}{\partial x_i} + \frac{\partial J}{\partial f_{i,y_i}} \cdot \frac{\partial f_{i,y_i}}{\partial x_i} \tag{2.10}$$

$$= \sum_j \frac{\partial J}{\partial f_{i,j}} \cdot w_j + \frac{\partial J}{\partial f_{i,y_i}} \left(\frac{\partial f_{i,y_i}}{\partial x_i} - w_{y_i} \right) \tag{2.11}$$

Аналогічно виконується для w:

$$\frac{\partial |w|}{\partial w} = \frac{w}{|w|}$$

$$\frac{\partial cos\theta}{\partial w} = \frac{\partial}{\partial x} \left(\frac{w^T x}{|w||x|} \right) = \frac{x}{|x||w|} - \frac{(w^T x)w}{|x||w|^3}$$

$$\frac{\partial sin^2\theta}{w} = \frac{\partial}{\partial w} \left(1 - cos^2\theta \right) = -2cos\theta \frac{\partial cos\theta}{\partial w}$$

$$cos(m\theta) = \sum_{n=0}^{\lfloor \frac{m}{2} \rfloor} (-1)^n \binom{m}{2n} (cos\theta)^{m-2n} (sin^2\theta)^n$$

$$\frac{\partial cos(m\theta)}{\partial w} = m(cos\theta)^{m-1} \frac{\partial cos\theta}{\partial w} + \tag{2.12}$$

$$+ \sum_{n=1}^{\lfloor \frac{m}{2} \rfloor} (-1)^n \binom{m}{2n} [n(cos\theta)^{m-2n} (sin^2\theta)^{n-1} \frac{\partial sin^2\theta}{\partial w} + \tag{2.13}$$

$$+ (m - 2n)(cos\theta)^{m-2n-1} (sin^2\theta)^n \frac{\partial cos\theta}{\partial w}] \tag{2.14}$$

$$f = (-1)^k |w||x| cos(m\theta) - 2k|w||x| = \left[(-1)^k cos(m\theta) - 2k \right] |w||x|$$

$$\frac{\partial f}{\partial w} = \left[(-1)^k cos(m\theta) - 2k \right] \frac{|x|}{|w|} w + (-1)^k |w||x| \frac{\partial cos(m\theta)}{\partial w}$$

$$\frac{\partial J}{\partial w_j} = \sum_{i,y_i \neq j} \frac{\partial J}{\partial f_{i,j}} \cdot \frac{\partial f_{i,j}}{\partial w_j} + \sum_{i,y_i=j} \frac{\partial J}{\partial f_{i,j}} \cdot \frac{\partial f_{i,j}}{\partial w_j} \tag{2.15}$$

$$= \sum_{i} \frac{\partial J}{\partial f_{i,j}} \cdot x_i + \sum_{i,y_i=j} \frac{\partial J}{\partial f_{i,j}} \cdot \left(\frac{\partial f_{i,j}}{\partial w_j} - x_i \right) \tag{2.16}$$

Отже, перетренована нейронна мережа з архітектурою MNIST та функцією втрат LSoftmax, яка була розрахована вище, розділяє на класи більш якісно, що свідчить про генералізацію моделі. Відстань між класами збільшується, як можна побачити на рисунку 2.3, що призводить до більшої точності моделі.

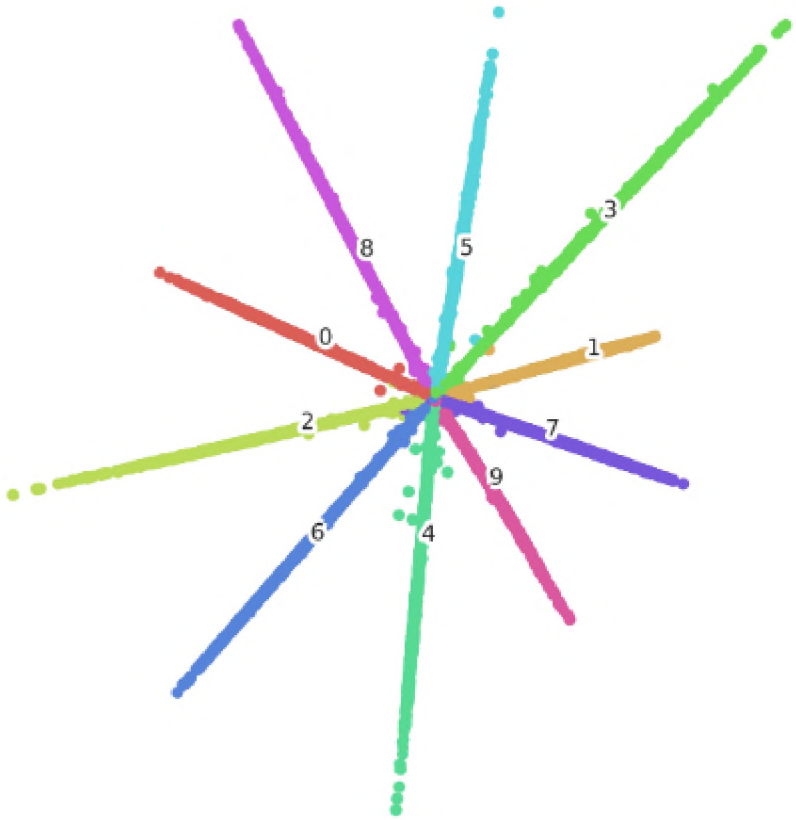

Рисунок 2.3 – Приклад розбиття ембедінгів на датасеті MNIST з LSoftmax функцією втрат та їх 2D t-SNE візуалізація

Висновки до розділу 2

У даному розділі було розглянуто основні функції втрат, а також функцію втрат LSoftmax[21]. Останню було докладно розібрано та були пророблені математичні перетворення для подальшого програмування цієї функції на мові Python. Використання LSoftmax функції втрат під час навчання дає змогу моделі більш чітко розуміти різницю між класами, що призводить до мінімізації помилки та покращення точності прогнозування. Це впливає також на застосування вкладень векторів такої моделі. Вона стає менш чутливою до відтінків, якщо подібні класи були додані до вибірки, а також до фону, яким є патерн футбольного поля.

3 РОЗРОБКА МОДЕЛІ НЕЙРОННОЇ МЕРЕЖІ

3.1 Засоби розробки програмного забезпечення

За останні роки виникла ціла екосистема бібліотек, в яких реалізовані найбільш відомі базові алгоритми машинного навчання. Окрім цього спільнота намагається програмувати всі останні дослідження та викладати у вільний доступ. Задля даної роботи було проаналізовано низку різних мов програмування, відкритих у вільному доступі розробок фреймворків та бібліотек.

Для програмної реалізації даної моделі було обрано мову програмування Python. Саме на базі цієї мови реалізована велика кількість бібліотек, які мають детальну документацію, надають змогу швидко опрацьовувати дані та в зручному виді використовувати алгоритми машинного навчання. Протягом останніх років Python набула величезної аудиторії та посіла почесне перше місце серед інших мов програмування у темпах росту. Детальніше вказано на Рисунку 3.1

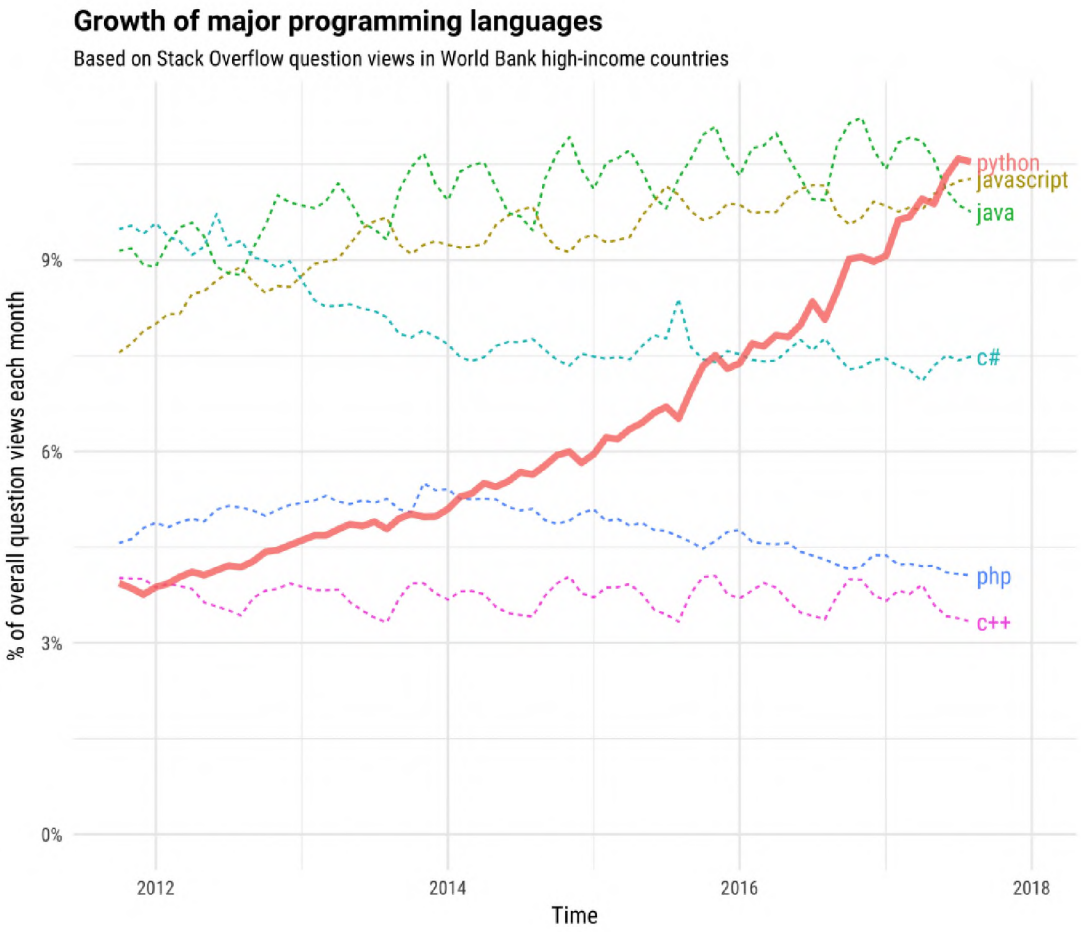

Рисунок 3.1 – Ріст популярності мов за даними stackoverflow.

Основою більшості бібліотек є NumPy, яка дозволяє ефективно працювати з числовими даними, матрицями, таблицями чисел і швидко проводити велику кількість типових операцій. Векторизація коду допомагає економити час як при навчанні, так і при вбудовуванні моделей у програми.

Для розробки моделі нейронної мережі було розглянуто декілька фреймворків, а саме: mxNet, tensorflow, pytorch та keras, який є надбудовою над tensofrlow та спрощує роботу з нейронними мережами. Порівняння останніх двох показано у Таблиці 3.1. Інші два фреймворки здалися менш зручними під час базового прототипування, тому вони не розглядалися більш детально для подальшої розробки моделі.

Порівняння фреймворків призвело до вибору pytorch в якості основного, оскільки розробка функцій втрат, які описувалися у розділі 2,

Таблиця 3.1 – Порівняння фреймворків keras та pytorch

	keras	pytorch
Переваги	- велика спільнота; - велика кількість різних прикладів; - простіша імплементація у додатки; - інтуїтивність; - велика кількість допоміжних бібліотек та модулів; - підтримка великої кількості різноманітних бекендів.	- інтуїтивність для Python розробників; - гнучкість у розробці персональних модулів; - лідуючий фреймворк з прототипування концептів; - висхідна наукова спільнота (особливо Китай); - наявність онлайн форуму з можливістю спілкуватися з розробниками напряму; - швидкість завдяки підтримки додатків на C++; - можливість дебагінгу та пошуку слабких місць.
Недоліки	- гнучкість; - наукова спільнота; - залежність від обраного бекенду.	- можливість використання у додатках (до публікації повного релізу); - документація; - кількість реалізованих проектів.

потребували саме гнучкості та можливості швидко прототипувати на мові Python.

3.2 Формування та обробка даних

Збір та попередня обробка датасету потребували кропіткої роботи з відеорядом футбольного матчу. Автоматичне формування вибірки майже не розглядалося, оскільки задача потребує знаходження об'єкту кластеризації в повній мірі на зображенні, яке буде приймати на вхід нейронної мережі. Оскільки цим об'єктом кластеризації є саме футболіст будь-якої з команд, то відсутність на зображенні ніг, рук або голови могло би призвести до засмічення даних, що, у свою чергу, призвело б до погіршення точності кластеризації. Тобто об'єкти повинні цілком вміщатися в деяких обмежених зонах, або граничних рамках. Приклад ідеального випадку розмітки зображено на Рисунку 3.2. Проте за умови такої розмітки була б і можливість використовувати автоматичну

розмітку даних. Оскільки подібних даних знайти у вільному доступі не вдалося, то було розроблено додаток, який дозволяє в ручному режимі збирати та розмічати датасет.

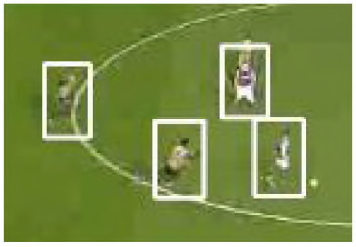

Рисунок 3.2 – Виділенні футболісти в граничних рамках

Додаток було розроблено за допомогою бібліотеки openCV, яка є лідуючим інструментом для прототипування у сфері комп'ютерного зору. Також вона надає можливість програмувати базовий графічний інтерфейс та підтримує низку базових функцій для обробки відео. Але, оскільки задача належить до типу задач з контрольованим навчанням, то функціонал додатку було розширено до можливості розмітки як позитивних класів (тобто футболіст, цілком розміщений у граничній рамці), так і негативних (тобто, поле, розмітка поля, трибуни тощо). Проте для задачі кластеризації та розбиття на команди достатньо лише позитивних класів, розмічених за кольором форми футболіста.

В загальній кількості було розмічено 29488 зображень футболістів на 26 різних класів. Класи вибирались випадково, в залежності від доступних у відкритому доступі відео матчів з панорамним видом. Розмічались в залежності від найпоширеніших кольорів форми: іноді це були футболка та шорти одного кольору, іноді — різного, навіть були футболка та смуги на ній, що різко виділяються. Перелік класів: жовтий (1760 зображень), червоний-білий (3269 зображень), білий-червоний (640 зображень), синій-чорний (816 зображень), жовтий-чорний (1331 зображення), зелений-білий (697 зображень), білий-чорний (523 зображення), чорний (1311 зображень), зелений (1715 зображень), жовтий-синій (730 зображень), фіолетовий-червоний (800 зображень),

білий-зелений (366 зображень), червоний (2718 зображень), білий-чорна-смуга-чорний (993 зображення), помаранчевий (524 зображення), голубий-чорний (518 зображень), жовтий-білий (312 зображень), фіолетовий (1506 зображень), червоний-синій (982 зображення), синій-білий (727 зображень), синій (1030 зображень) та помаранчевий-чорний (938 зображень). Приклади класів зображено на Рисунку 3.3

Рисунок 3.3 – Вирізані футболісти, які розмічені різними класами

Отримані граничні рамки за розміром коливалися біля 40x25 по висоті та ширині відповідно. Така низька розмірність зумовлена панорамним відеорядом, з якого збирався датасет.

3.3 Архітектура нейронної мережі

3.3.1 Основна архітектура

Основою архітектурної мережі є вихідний код вирішення задачі датасету MNIST. Архітектура складалась з чотирьох конволюційних блоків, активаційної функції ReLU, яка зображена на рисунку 3.4, і батч-нормалізації. Проте метрика точності на валідаційному датасеті досягала не досить великої позначки, приблизно 0.912. Тому для покращення цих результатів було розглянуто деякі сучасні шари та архітектури нейронних мереж, а саме CoordConv, SqueezeNet та PNN.

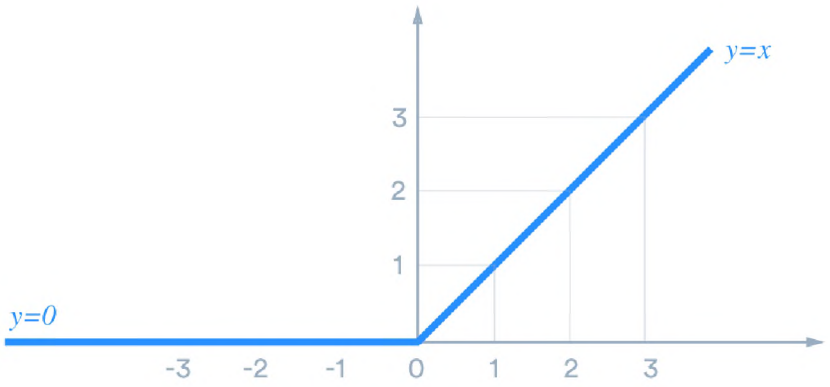

Рисунок 3.4 – Функція активації ReLU

3.3.2 SqueezeNet

SqueezeNet — це приклад архітектури, запропонований Йандола та ін.[23], який обіцяє трикратний приріст швидкості та п'ятисот кратне зменшення розміру нейронної мережі, у порівнянні з класичною сьогодні архітектурою AlexNet. Досягнення цих результатів пропонується не лише додаванням шарів у нейронну мережу, але ще й зміною кількості фільтрів у класичних згорткових блоках.

Головними ідеями є:

1) використання 1x1 (точкових) фільтрів, щоб замінити 3x3 фільтри; як перший, тільки 1/9 обчислень;

2) використання 1x1 фільтрів як шар вузьких місць, щоб зменшити глибину для зменшення обчислення наступних 3x3 фільтрів;

3) зменшення розмірності вибірки для збереження великої кількості ознак.

Заміна шарів підвищила точність до 0.974, а також стиснуло розмір отриманої моделі до кількох сотень кілобайтів. Проте розмір не є принциповим, оскільки результат не займає багато місця та дозволяє транспортувати модель навіть на кишенькові пристрої.

3.3.3 Пертурбаційні нейронні мережі

Схожий підхід запропонували Юафей-Ксу та ін. у своїй роботі про Пертурбаційні нейронні мережі (або PNN)[24]. Їх ідея полягає в тому, щоб замінити 3x3 згортку на 1x1 додаючи деякий шум на вхід. Приклад роботи зображено на рисунку 3.5. Проте автори допустили помилку у першій версії своєї роботи, яка була представлена на конференції CVPR 2018 року – не вірно розрахували точність під час перевірки. Це призвело до втрати всіх результатів. Тестування їх роботи для вирішення задачі спочатку також не показувало кращих результатів. Проте за підтримки Майкла Клачко, вони виклали у відкритий доступ оновлену версію[25], яка показала точність 0.990. Але ініціалізація шуму займає дуже багато часу, що гальмує роботу моделі у 12 разів. Отже, така архітектура не задовольняє поставленим вимогам і не може розцінюватися як покращення класичної моделі.

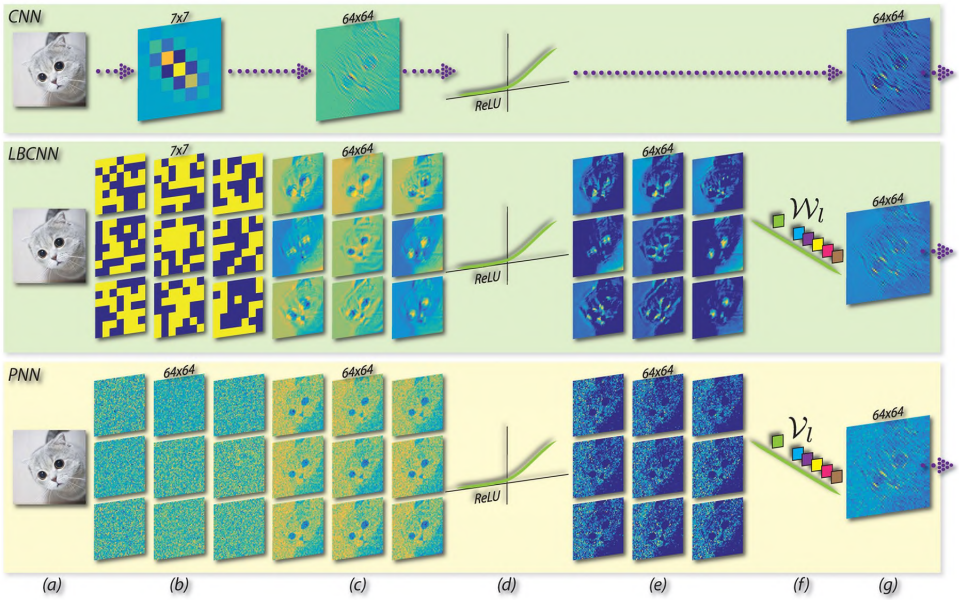

Рисунок 3.5 – Приклад роботи PNN в порівнянні з іншими архітектурами з оригінальної публікації[24]

3.3.4 CoordConv шар

Ліу, Ремен та ін. у 2018 році випустили наукову роботу під назвою "Інтригуюча невдача конволюційних нейронних мереж та рішення CoordConv"[26]. Вони запропонували покращення роботи фільтрів за допомогою додання ще двох: i та j координат. Приклад додавання фільтрів зображено на рисунку 3.6.

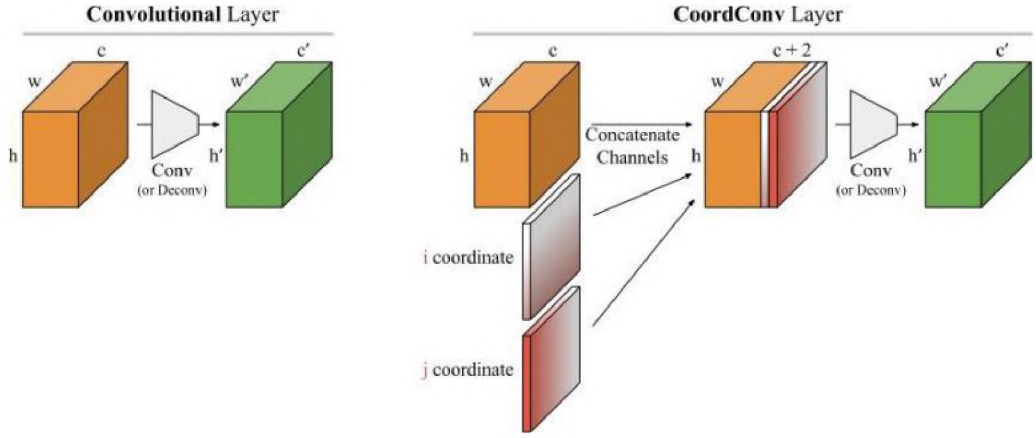

Рисунок 3.6 – Приклад аугментації конволюційних фільтрів та отримання CoordConv шару

КНН отримали успіх завдяки деяким важливим факторам: використанню відносно невеликої кількості параметрів навчання, швидкості обчислювання на сучасних графічних процесорах і навчання на еквіваріантній функції.

CoordConv зберігає перші дві з цих властивостей. Але якщо ваги з координат під час навчання обертаються нулем, CoordConv поводитися як стандартна згортка. Крім того, як зазначають автори, моделі CoordConv мають в 10-100 разів менше параметрів, тренуються за секунди, а не за годину (у 150 разів швидше), як це і очікується від найефективніших стандартних CNN.

У реальному випадку, даний шар був переписаний на фреймворку pytorch і його швидкість роботи нижча, ніж у класичному випадку або за

використання SqueezeNet. Проте вона задовольняє умові в 0.04 секунди. А точність досягла значення в 0.984.

3.4 Контекстне вдосконалення результатів

Додавання зовнішніх (оскільки вони не вбудовані в обраний фреймворк) шарів набагато збільшило точність прогнозування нейронної мережі. Але використання контекстної інформації під час тренування також сприяло покращенню результатів.

Найбільше впливу мало додавання трансформації даних: як до тренувальної, так і валідаційної вибірки. Загальною для обох була випадкова зміна яскравості. Інші відрізнялись або зміною параметрів, або зовсім не використовувались. Трансформації тренувальної вибірки містили: випадкове дзеркальне відображення по горизонталі та випадкове обертання під кутом 30 градусів. До валідаційної належали: випадкове обертання під кутом 7 градусів та випадкові афінні перетворення — зсув зображення на 10 відсотків по вертикалі.

Усі ці трансформації пояснені специфікою зібраних даних. Наприклад, яскравість зумовлена зміною освітлення на полі. Тобто, частіше за все поле знаходиться під голим небом, що призводить до утворення тіні. Дані збирались протягом усієї гри, тому футболісти досить часто потрапляли на неосвітлені площі. Випадкова зміна яскравості дозволила виправити цю ситуацію і зробити нейронну мережу менш чутливою до перепадів світла. Тренувальна вибірка була модифікована двома способами. Перший – випадкове дзеркальне відображення – дозволило зменшити чутливість до тайму, тобто до сторони з якої грають гравці, а другий, обертання, – чутливість до пози, в якій знаходиться гравець, наприклад, під час бігу. Валідаційна вибірка також була модифікована за допомогою внесення трансформації обертання зображення, але під іншим кутом, меншим. Це зумовлено

бажанням внесення "шуму"до даних. З цієї ж причини було додано зсув на 10 відсотків, що малювало чорну смугу з однієї зі сторін зображення, а також зсувало об'єкт, футболіста, з центра картинки.

Висновки до розділу 3

У даному розділі було проаналізовано та описано основні засоби розробки моделі даної роботи. Було порівняно два найбільш популярних і зручних фреймворки для побудови нейронних мереж, в результаті якого було обрано фреймворк pytorch. Використання описаних бібліотек відчутно спростило етапи написання додатку для збору даних, їх попередньої обробки та побудови моделі нейронної мережі.

Серед низки вже розроблених архітектур, які є доступні у вільному доступі, а саме архітектура для вирішення задачі датасету MNIST послугувала базою для побудови моделі нейронної мережі. Сама архітектура була модифікована шляхом дописаних шарів для отримання кращої точності кластеризації.

Також було використано трансформації вибірки, які ще називаються аугментацією даних [27]. Це допомогою збільшити точність прогнозування без втрат швидкості роботи моделі, оскільки тренування нейронної мережі відбувається до роботи програми, тобто до процесу прогнозування моделі.

ВИСНОВКИ

У результаті дослідження вдалось розробити модель автоматичної кластеризації гравців на футбольному полі. Програма автоматично розбиває задані об'єкти з досить великою точністю. Значення метрики досягли позначки в 0.984 на валідаційній вибірці, яка складалась з меншої кількості класів, що використовуються при тренуванні, а також зібрані з ігор команд, які не враховувались у тренувальній вибірці. Точності з використанням різних архітектур та аугментацією даних показано у таблиці 3.2.

Приклад результатів роботи моделі зображено на рисунках 3.7 та 3.8. На рисунку 3.7 виділено всіх персон на полі: бокових суддів, центрового суддю та гравців обох команд. Класи було розподілено наступним чином: воротар першої команди, перша команда, воротар другої команди, друга команда та судді (два бокових та центровий). На рисунку 3.8 зображено розбитих на класи всіх персон. Також, на другому рисунку можна побачити виділений м'яч і тренера команди, але вони позначені сторонніми класами та не враховувались під час кластеризації.

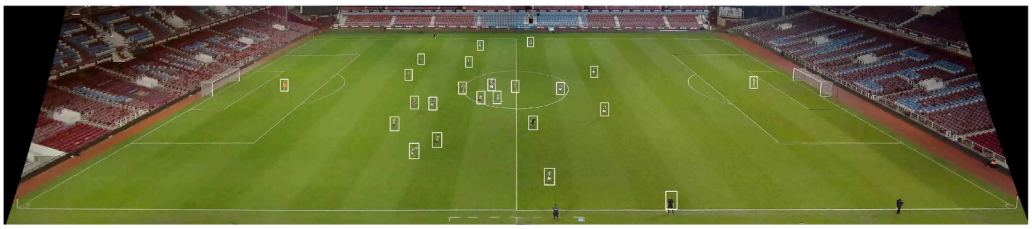

Рисунок 3.7 – Кадр з футбольного матчу з виділеними персонами на полі в якості 4K

Таблиця 3.2 – Порівняння результатів метрики точності при різних архітектурах та відповідності до поставленої мінімальної границі швидкості відпрацювання програми.

Архітектура	Точність (Accuracy)	Робота в режимі реального часу
Класична для задачі датасету MNIST та крос-ентропія	0.912	так
Класична для задачі датасету MNIST та LSoftmax	0.933	так
Модифікована з PNN та LSoftmax	0.990	ні
Модифікована з CoordConv та LSfotmax	**0.984**	**так**
Модифікована з SqueezeNet та LSoftmax	0.974	так

Рисунок 3.8 – Кадр з футбольного матчу з виділеними та кластеризованими персонами на полі в якості 4K

Модель була тестована разом з програмою автоматичного пошуку гравців на футбольному полі, яка є власністю компанії SoftConstruct Україна, і працює в режимі реального часу. Тобто, на кластеризацію витрачається менше, ніж 0.04 секунди, що дозволяє використовувати модель при передачі відео потоку в якості 4K.

Створена модель може полягати подальшій оптимізації та вдосконаленню. Оскільки для розробки використовувався фреймворк pytorch версії 0.4.1, то з виходом новішої версії буде можливість використовувати додаткові модулі з мови програмування C++, що, у свою чергу, дозволить експериментувати з іншими шарами нейронних мереж, які є більш вимогливими. Також при невеликих змінах архітектури та тренувальної вибірки, модель може бути використана для

кластеризації за кольором будь-якої кількості класів інших об'єктів на зображеннях.

ПЕРЕЛІК ПОСИЛАНЬ

1. A. Rabinovich. Then, Now, Tomorrow: Neural Networks for Computer Vision [Виступ з конференції та слайди] – https://eecvc.com/eecvc-2017-slides-and-videos/ – 2017.

2. CSRankings: Computer Science Rankings. – http://csrankings.org/#/index?all – 2016-2018.

3. B. Schiele. Progress in Computer Vision in the Last Decade Open Problems: People Detection & Human Pose Estimation. – http://machinescansee.com/wp-content/uploads/2018/07/schiele-moskau-v1.pdf – 2018.

4. Mikolov, Tomas; Sutskever, Ilya; Chen, Kai; Corrado, Greg; Dean, Jeffrey. Distributed Representations of Words and Phrases and their Compositionality. – arXiv:1310.4546. – 2013.

5. Shai Shalev-Shwartz and Shai Ben-David. Understanding Machine Learning: From Theory to Algorithms. // *Cambridge University Press.* – 2014. – p.258.

6. Jeffrey Pennington, Richard Socher, Christopher D. Manning. GloVe: Global Vectors for Word Representation. – nlp.stanford.edu/projects/glove/ – 2015.

7. Laurens Van Der Maaten. t-SNE [Персональний блог] – https://lvdmaaten.github.io/tsne/

8. Laurens van der Maaten. Visualizing Data using t-SNE //*Journal of Machine Learning Research.* – 2008. – http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf

9. Pearson, K. On Lines and Planes of Closest Fit to Systems of Points in Space. // *Philosophical Magazine. 2 (11).* – 1901. – pp.559–572. – doi:10.1080/14786440109462720

10. Sangoh Jeong. Histogram-Based Color Image Retrieval. // *Psych221/EE362 Project Report.* – 2001. – March. – https://pdfs.semanticscholar.org/e884/2a22aa486fd273606fcd5f8090619bbb468c.pdf

11. Wikipedia. ArrayFire [Вільна енциклопедія] – https://en.wikipedia.org/wiki/ArrayFire

12. F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain // *Psychological Review – Cornell Aeronautical Laboratory.* – 1958. – Vol. 65, No. 6. – http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf

13. Andrej Karpathy and Justin Johnson and Fei-Fei Li. Visualizing and Understanding Recurrent Networks. - *CoRR.* - abs/1506.02078. - 2015. - http://dblp.uni-trier.de/rec/bib/journals/corr/KarpathyJL15

14. David Stutz. Understanding Convolutional Neural Networks // *Seminar Report* – 2014. – August. – https://davidstutz.de/wordpress/wp-content/uploads/2014/07/seminar.pdf

15. N Bartneck et al. Color segmentation with polynomial classification // 0-8186-2915-0/92. – 1992. – pp.635-638.

16. Carolanne Mangles. Gartner Hype Cycle 2018 – Most emerging technologies are 5-10 years away // *SmartInsights.* – 2018. – August. – https://www.smartinsights.com/managing-digital-marketing/managing-marketing-technology/gartner-hype-cycle-2018-most-emerging-technologies-are-5-10-years-away/

17. Aretz, Kevin; Bartram, Söhnke M.; Pope, Peter F. Asymmetric Loss Functions and the Rationality of Expected Stock Returns. // *International Journal of Forecasting 27 (2).* – 2011. – April–June. – pp.413–437. doi:10.1016/j.ijforecast.2009.10.008

18. Goldberger, Arthur S. Classical Linear Regression // *Econometric Theory. New York: John Wiley & Sons.* – 1964. – pp. 156–212. – ISBN 0-471-31101-4.

19. Plunkett and Elman. Exercises in Rethinking Innateness // *The MIT Press.* – 1997. – p.166

20. Singhal, Amit. Modern Information Retrieval: A Brief Overview. // *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24 (4).* – 2001. – pp.35–43.

21. Weiyang Liu et al. Large-Margin Softmax Loss for Convolutional Neural Networks – arXiv:1612.02295. – 2017. – November.

22. Weiyang Liu. Large-Margin Softmax Loss for Convolutional Neural Networks [Репозиторій з проектом] – 2018. – https://github.com/wy1iu/LargeMargin_Softmax_Loss

23. Forrest N. Iandola et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. – arXiv:1602.07360 – 2016. – November.

24. Felix Juefei-Xu et al. Perturbative Neural Networks // *CVPR 2018.* – arXiv:1806.01817. – 2018. – June.

25. Felix Juefei-Xu. New PNN Repository. [Репозиторій з проектом] – 2018. – https://github.com/juefeix/pnn.pytorch.update

26. Rosanne Liu, Joel Lehman et al. An Intriguing Failing of Convolutional Neural Networks and the CoordConv Solution. // *NeurIPS 2018.* – arXiv:1807.03247. – 2018. – December.

27. David A. Van Dyk, Xiao-Li Meng. The Art of Data Augmentation // *Journal of Computational and Graphical Statistics (1).* – 2001. – Vol. 10. – pp.1-50. – http://www.stat.harvard.edu/Faculty_Content/meng/JCGS01.pdf

ДОДАТОК А

Лістинг коду програми для навчання нейронної мережі

```
import argparse
from pathlib import Path
from time import time

import torch
import torch.backends.cudnn as cudnn
import torchvision.transforms as transforms
import yaml
from torch import nn
from torch.nn.utils import clip_grad_norm_
from torch.optim import Adam, lr_scheduler
from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder

from .networks import (ball_classifier, half_field_line_detector,
                       line_detector, person_classifier, team_classifier)
from .utils import to_bgr_transform, polynomial_lr
from .scale_and_split_ball_dataset import scale_and_split


def get_transform(model_name, data_type_dir):
    default_transform = [
        transforms.ToTensor(),
        transforms.Lambda(to_bgr_transform)
    ]
    mode_transform = []
    if "test" in data_type_dir:
        pass
    elif "line" in model_name and "train" in data_type_dir:
        mode_transform += [
            transforms.ColorJitter(brightness=0.1),
            transforms.RandomHorizontalFlip(),
            transforms.RandomRotation(30)
        ]
    else:
        mode_transform += [
            transforms.RandomHorizontalFlip(),
            transforms.RandomRotation(7),
            transforms.RandomAffine(0, translate=(0.1, 0.1))
        ]
    return transforms.Compose(mode_transform + default_transform)


def get_data_loader(img_path, data_type_dir, model_name, **kwargs):
    transform = get_transform(model_name, data_type_dir)
    dataset = ImageFolder(img_path / data_type_dir, transform=transform)
    return DataLoader(dataset, **kwargs)


def parse_arguments():
```

```
51      parser = argparse.ArgumentParser()
52      parser.add_argument(
53          "-m", "--model_name", required=True, type=str,
54          choices=["person", "team", "ball", "line", "half_field_line"],
55          dest="model_name", help="set␣model_name␣to␣train␣appropriate␣model")
56      args = parser.parse_args()
57      return args.model_name
58
59
60  def forward_step(x, y, model, criterion, model_name, training=True, device="cpu"):
61      x = torch.tensor(x).to(device)
62      y = torch.tensor(y).to(device)
63
64      if model_name == "team":
65          output = model(x, y)[0] if training else model(x)[0]
66      else:
67          output = model(x)[1]
68
69      loss = criterion(input=output, target=y).to(device)
70      y_pred = output.data.max(1)[1] if not training else None
71      return x, y, loss, y_pred
72
73
74  class Config:
75      def __init__(self, model_name):
76          self.model_name = model_name
77          with open("config/train.yml") as f:
78              self.config = yaml.safe_load(f)
79
80      def get(self, key):
81          try:
82              return self.config[self.model_name][key]
83          except KeyError:
84              return self.config["common"][key]
85
86
87  def run():
88      print("STATUS␣|␣Data␣preprocessing␣...")
89
90      model_name = parse_arguments()
91      config = Config(model_name=model_name)
92
93      # Train/Validation data split
94      train_dir = config.get("train_dir")
95      test_dir = config.get("test_dir")
96      image_channels = config.get("image_channels")
97      image_path = Path(str(Path(config.get("image_path"))))
98      if "ball" in model_name and not image_path.exists():
99          scale_and_split(config.get("verbose_scale_and_split"))
100
101     # Load model
102     model_parameters = {"in_channels": image_channels}
103     module_from_model_type = {
104         "person": person_classifier, "line": line_detector, "half_field_line": half_field_line_detector,
```

```
105         "ball": ball_classifier, "team": team_classifier}
106     module = module_from_model_type[model_name]
107     if model_name == "team":
108         num_classes = sum(subdir.is_dir() for subdir in (image_path / train_dir).iterdir())
109         model_parameters.update({"out_channels": num_classes, "margin": config.get("margin")})
110 
111     model = module.Net(**model_parameters)
112     device = model.device
113     print("INFO␣␣␣|␣Device:", device.type)
114     learning_rate = config.get("learning_rate")
115     criterion = nn.CrossEntropyLoss().to(device)
116     optimizer = Adam(model.parameters(), lr=learning_rate, weight_decay=0.00005)
117 
118     def lambda_lr(epoch):
119         return polynomial_lr(learning_rate, epoch, max_epoch=config.get("num_epochs"), power=0.4)
120     learning_rate_scheduler = lr_scheduler.LambdaLR(optimizer, lr_lambda=lambda_lr)
121 
122     # Prepare dataloaders with defined batch size
123     batch_size = config.get("batch_size")
124     pin_memory = False if device.type == "cpu" else True
125     train_loader = get_data_loader(
126         image_path, train_dir, model_name, batch_size=batch_size, shuffle=True, num_workers=8, pin_memory=pin_memor
127     validation_loader = get_data_loader(
128         image_path, test_dir, model_name, batch_size=batch_size, shuffle=True, num_workers=8, pin_memory=pin_memory
129 
130     # Decrease training time
131     cudnn.benchmark = True
132     # Make results more deterministic
133     cudnn.deterministic = True
134 
135     def training(epoch):
136         learning_rate_scheduler.step()
137         model.train()
138         total_loss = 0
139         mini_batch_step = config.get("mini_batch_step")
140 
141         for batch_step, (x, y) in enumerate(train_loader):
142             optimizer.zero_grad()
143 
144             x, y, loss, _ = forward_step(x, y, model, criterion, model_name, True, device)
145             loss.backward()
146 
147             clip_grad_norm_(model.parameters(), max_norm=10)
148             optimizer.step()
149 
150             total_loss += loss.item()
151             if batch_step % mini_batch_step == mini_batch_step - 1:
152                 mini_batch_loss = total_loss / mini_batch_step
153                 print(
154                     f"STATUS␣|␣Train␣Epoch:␣{epoch}␣[{batch_step␣*␣len(x)}/{len(train_loader.dataset)}␣"
155                     f"({100␣*␣batch_step␣/␣len(train_loader):.0f}%)]\tLoss:␣{mini_batch_loss:.6f}"
156                 )
157                 total_loss = 0
158 
```

```
159     @torch.no_grad()
160     def validation(epoch):
161         # TODO: move metrics to decorator function
162         model.eval()  # Reqired parameter in validation mode
163         num_correct = total_loss = 0
164         num_correct_tp = num_correct_fp = 0
165         pos_true = neg_true = 0
166         for _, (x, y_true) in enumerate(validation_loader):
167             x, y_true, loss, y_pred = forward_step(x, y_true, model, criterion, model_name, False, device)
168 
169             total_loss += loss.item()
170             num_correct += y_pred.eq(y_true).long().sum().item()
171 
172             pos_true_batch = sum(y_true == 1).item()
173             pos_true += pos_true_batch
174             neg_true += len(y_true) - pos_true_batch
175             num_correct_tp_batch = y_pred[y_true == 1].eq(y_true[y_true == 1]).long().sum().item()
176             num_correct_tp += num_correct_tp_batch
177             num_correct_fp += y_pred[y_pred == 1].long().sum().item() - num_correct_tp_batch
178 
179         average_loss = total_loss / len(validation_loader)
180         accuracy = num_correct / len(validation_loader.dataset) * 100
181 
182         print(
183             f"\n"
184             f"INFO␣␣␣|␣Test␣set:\tLoss:␣{average_loss:.6f}␣\n"
185             f"INFO␣␣␣|␣Metrics:␣\tAccuracy:␣{num_correct}/{len(validation_loader.dataset)}␣({accuracy:.4f}␣%)")
186         if model_name != "team":
187             accuracy_tp = num_correct_tp / pos_true * 100
188             accuracy_fp = num_correct_fp / neg_true * 100
189             print(
190                 f"INFO␣␣␣|␣\t\tTP:␣{num_correct_tp}␣({accuracy_tp:.4f}␣%)\n"
191                 f"INFO␣␣␣|␣\t\tFP:␣{num_correct_fp}␣({accuracy_fp:.4f}␣%)\n")
192 
193     start_time = time()
194 
195     print("STATUS␣|␣Model␣training␣and␣validation␣...\n")
196     # NOTE: first epoch will be skipped
197     for epoch in range(1, config.get("num_epochs")):
198         training_start_time = time()
199         training(epoch)
200         training_end_time = time()
201         validation(epoch)
202         print(f"INFO␣␣␣|␣Validation␣pass␣time:␣\t{time()␣-␣training_end_time}␣seconds")
203         print(f"INFO␣␣␣|␣Epoch␣{epoch}␣pass␣time:␣\t{time()␣-␣training_start_time}␣seconds\n")
204 
205     # NOTE: turns on unsafe save mode
206     print("\nSTATUS␣|␣Saving␣model␣...")
207     model_name = config.get("model_name") + ".pytorch.model"
208     model_path = Path(config.get("model_save_dir")) / model_name
209     torch.save(model, model_path)
210     print(f"INFO␣␣␣|␣Model␣saved␣successfully")
211     print(f"INFO␣␣␣|␣Training␣time:␣{time()␣-␣start_time:.1f}␣s")
212 
```

```
213
214  if __name__ == "__main__":
215      run()
```

ДОДАТОК Б

Лістинг коду програми для ручної розмітки даних

```
import argparse
import os
import random
import uuid

import cv2

# Global settings
image_prefix = 'img'
window_name = 'frame'
drawing = False
x0, y0 = -1, -1
img_height = 40
img_width = 25


def box_contain_point(box, point):
    x, y = point
    x_top, y_top, w, h = box
    return True if x_top < x < x_top + w and y_top < y < y_top + h else False


def click_and_crop_pos(event, x, y, flags, param):
    global drawing, x0, y0
    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        x0, y0 = x, y
    elif event == cv2.EVENT_MOUSEMOVE and drawing:
        cv2.rectangle(img=frame_output, pt1=(x0, y0), pt2=(
            x, y), color=(192, 192, 192), thickness=1)
        cv2.addWeighted(frame_copy, 0.3, frame_output,
                        0.7, 0, frame_output)
        cv2.imshow(window_name, frame_output)
    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
        cv2.rectangle(img=frame_output, pt1=(x0, y0), pt2=(
            x, y), color=(0, 255, 0), thickness=2)
        cv2.imshow(window_name, frame_output)
        cropped_image = frame[y0:y, x0:x]
        cropped_image = cv2.resize(cropped_image, (img_width, img_height))
        uid = str(uuid.uuid4())
        cv2.imwrite(f"{save_folder}{image_prefix}_{uid}.jpg", cropped_image)


def click_and_crop_neg(event, x, y, flags, param):
    if event != cv2.EVENT_LBUTTONDOWN:
        return
    for n in range(num_neg_samples):
        uid = str(uuid.uuid4())
        xs = random.randint(x - radius, x + radius)
```

```
51          ys = random.randint(y - radius, y + radius)
52          cropped_image = frame[ys - y_step:ys +
53                                y_step, xs - x_step:xs + x_step]
54          cropped_image = cv2.resize(cropped_image, (img_width, img_height))
55          cv2.imwrite(f"{save_folder}{image_prefix}_{uid}.jpg", cropped_image)
56
57
58  if __name__ == '__main__':
59      parser = argparse.ArgumentParser()
60      parser.add_argument('-m', required=True, choices=['pos', 'neg'], dest='mode',
61                          help='Mode to crop positive or negative class of images')
62      parser.add_argument('-i', required=True, dest='input',
63                          help='Path to video file')
64      parser.add_argument('-o', required=True, dest='output',
65                          help='Path to output folder')
66      parser.add_argument('--y_step', required=False, type=int, default=40,
67                          help='OY step shift size [default: 40]')
68      parser.add_argument('--x_step', required=False, type=int, default=25,
69                          help='OX step shift size [default: 25]')
70      parser.add_argument('--radius', required=False, type=int, default=20,
71                          help='Radius for random sampling [default: 20]')
72      parser.add_argument('--num_neg_samples', required=False, type=int, default=5,
73                          help='Number of random samples per click [default: 5]')
74      parser.add_argument('--frame', required=False, type=int, default=0,
75                          help='Start frame [default: 0]')
76      args = parser.parse_args()
77
78      mode = args.mode
79      video_file = args.input
80      save_folder = args.output
81      y_step = args.y_step
82      x_step = args.x_step
83      radius = args.radius
84      num_neg_samples = args.num_neg_samples
85      start_frame = args.frame
86
87      if not os.path.exists(save_folder):
88          os.makedirs(save_folder)
89
90      video = cv2.VideoCapture(video_file)
91      cv2.namedWindow(window_name)
92
93      if mode == 'neg':
94          cv2.setMouseCallback(window_name, click_and_crop_neg)
95      elif mode == 'pos':
96          cv2.setMouseCallback(window_name, click_and_crop_pos)
97
98      counter = 0
99      while True:
100         flag, frame = video.read()
101         if not flag:
102             break
103         frame_copy = frame.copy()
104         frame_output = frame.copy()
```

```
105
106         counter += 1
107         if counter > start_frame:
108             cv2.imshow(window_name, frame)
109             key = cv2.waitKey(0)
110             if key == ord('q'):
111                 break
112             else:
113                 pass
```

ДОДАТОК В

Лістинг коду архітектури нейронної мережі

```
from torch import nn

from .base import BaseNet, same_padding, valid_padding
from .layers import CoordConv, Flatten
from .losses import LargeMarginCosineLoss


class Net(BaseNet):
    def __init__(self, in_channels, out_channels, margin):
        self.margin = margin
        self.out_channels = out_channels
        super().__init__(in_channels)

    def arch(self):
        kernel_size = (1, 1)
        same = same_padding(kernel_size)
        valid = valid_padding(kernel_size)

        self.conv_block1 = nn.Sequential(
            CoordConv(self.in_channels, 16, radius_channel=True, kernel_size=kernel_size, padding=same),
            nn.PReLU(),
            nn.BatchNorm2d(16, affine=False))

        self.conv_block2 = nn.Sequential(
            CoordConv(16, 16, radius_channel=True, kernel_size=kernel_size, padding=valid),
            nn.PReLU(),
            nn.BatchNorm2d(16, affine=False),
            nn.MaxPool2d((2, 2), stride=2))

        self.conv_block3 = nn.Sequential(
            CoordConv(16, 32, radius_channel=True, kernel_size=kernel_size, padding=same),
            nn.PReLU(),
            nn.BatchNorm2d(32, affine=False))

        self.conv_block4 = nn.Sequential(
            CoordConv(32, 32, radius_channel=True, kernel_size=kernel_size, padding=valid),
            nn.PReLU(),
            nn.BatchNorm2d(32, affine=False),
            nn.MaxPool2d((2, 2), stride=2))

        self.dense_block = nn.Sequential(
            Flatten(),
            # 32 * 10 * 6 = out_conv_4 * ((in_height - k_size_1) / max_pool_1 - k_size_1) / max_pool_2 *
            #               * ((in_width - k_size_1) / max_pool_1 - k_size_1) / max_pool_2
            nn.Linear(32 * 10 * 6, 64),
            nn.BatchNorm1d(64, affine=False))

        self.margin = LSoftmaxLinear(in_channels=64, out_channels=self.out_channels, m=4)

    def forward(self, x, y=None):
```

```
51          out = self.conv_block1(x)
52          out = self.conv_block2(out)
53          out = self.conv_block3(out)
54          out = self.conv_block4(out)
55          out = self.dense_block(out)
56          logit = self.margin(out, y)
57          return logit, out
```

ДОДАТОК Г

Лістинг коду з додатковими функціями для програми тренування нейронної мережі

```
import os
import random
import shutil
from pathlib import Path

import cv2

from config import random_seed

IMG_EXTENSIONS = (".jpg", ".jpeg", ".png", ".ppm", ".bmp", ".pgm", ".tif")


def _sort_by_splitted_dirs(data_root, labeled_images, data_root_split_subdir):
    for image_class, image_name in labeled_images:
        _move_image(data_root, image_class, image_name, data_root_split_subdir)


def _move_image(data_root, image_class, image_name, data_root_split_subdir):
    original_image_path = data_root / image_class / image_name
    new_path_to_image = data_root / data_root_split_subdir / image_class
    new_path_to_image.mkdir(parents=True, exist_ok=True)
    shutil.move(original_image_path, new_path_to_image / image_name)


def get_images(data_root):
    """
    Get the list of images from all subdirs (classes) in image dir
    """
    all_images = []
    classes = set()
    for directory, _, img_names in os.walk(data_root):
        directory_name = Path(directory).name
        for img_name in img_names:
            if not img_name.lower().endswith(IMG_EXTENSIONS):
                continue
            classes.add(directory_name)
            all_images.append((directory_name, img_name))
    print("INFO␣␣␣|␣Number␣of␣Classes:", len(classes))
    return all_images, classes


def train_test_split_dir(data_root, test_ratio, train_dir, test_dir, image_shape):
    """
    A generic data sampler where the samples are arranged in this way:

                        data_root
                       /         /
                  train           test
                 / ... /         / ... /
```

```
50              class1  classN  class1  classN
51
52      Each subdir in `data_root` represents class (label) for classification problem;
53      `test_ratio` represents test set to train set ratio (or split ratio);
54      `train_dir` and `test_dir` represent output dir names.
55      """
56      if "splitted" in str(data_root):
57          return data_root
58      print("STATUS | Splitting data into train and test sets ...")
59
60      if (data_root / train_dir).exists() or (data_root / test_dir).exists():
61          raise OSError(
62              f"{train_dir} or {test_dir} dirs exist. Check the correctness "
63              "of specified data root dir or change class names")
64      if not 0 < test_ratio < 1:
65          raise ValueError("Test ratio should be in range from 0 to 1")
66
67      new_data_root = Path(str(data_root) + "_splitted")
68      if new_data_root.exists():
69          print("INFO    | Continue with preprocessed and stored data in", new_data_root)
70          return new_data_root
71
72      shutil.copytree(data_root, new_data_root)
73      all_images, all_classes = get_images(new_data_root)
74
75      print("STATUS | Checking data shape matching with image shape ...")
76      remove_counter = [
77          all_images.remove((img_class, img_name))
78          for img_class, img_name in all_images
79          if cv2.imread(str(data_root / img_class / img_name)).shape != image_shape]
80      print("INFO    | Files removed:", len(remove_counter))
81
82      print("STATUS | Splitting data ...")
83      random.seed(random_seed)
84      random.shuffle(all_images)
85      split_index = int(len(all_images) * (1 - test_ratio))
86      print("INFO    | Test split ratio:", test_ratio)
87      assert(split_index > 0)  # Otherwise, test_ratio is too small
88      train_imgs = all_images[:split_index]
89      print("INFO    | Training set size:", len(train_imgs))
90      test_imgs = all_images[split_index:]
91      print("INFO    | Test set size:", len(test_imgs))
92
93      # Move images to split subdirs
94      _sort_by_splitted_dirs(new_data_root, train_imgs, train_dir)
95      _sort_by_splitted_dirs(new_data_root, test_imgs, test_dir)
96
97      # Removes dirs with inappropriate data
98      for directory in all_classes:
99          shutil.rmtree(new_data_root / directory)
100
101     return new_data_root
102
103
```

```
104 def to_bgr_transform(tensor):
105     return tensor[[2, 1, 0], :, :]
106
107
108 def polynomial_lr(default_lr, epoch, max_epoch=100, power=0.9):
109     return default_lr * ((1 - float(epoch) / max_epoch) ** power)
```

ДОДАТОК Д

Лістинг коду з базовим класом нейронної мережі

```
import torch
from torch import nn

from config import random_seed

if torch.cuda.is_available():
    device = torch.device("cuda:0")
    torch.cuda.manual_seed_all(random_seed)
else:
    device = torch.device("cpu")
    torch.manual_seed(random_seed)


class BaseNet(nn.Module):
    device = device

    def __init__(self, in_channels):
        super().__init__()
        self.in_channels = in_channels
        self.arch()
        self.to(self.device, non_blocking=True, dtype=torch.float32)
        self.softmax = nn.Softmax(1)

    def arch(self):
        raise NotImplementedError

    def get_embeddings(self, x):
        return self.to_numpy(self.predict(x))

    def get_scores(self, x):
        return self.to_numpy(self.predict_proba(x))

    def to_numpy(self, tensor):
        data = tensor.data.cpu() if self.device.type != "cpu" else tensor.data
        return data.numpy()

    @torch.no_grad()
    def predict(self, x):
        x = torch.from_numpy(x).permute(0, 3, 1, 2).float().div(255).detach()
        return self(x.to(self.device, non_blocking=True, dtype=torch.float32))

    def predict_proba(self, x):
        return self.softmax(self.predict(x))


def same_padding(kernel_size):
    return tuple((k - 1) // 2 for k in kernel_size)


def valid_padding(kernel_size):
```

```
51        return tuple(0 for _ in kernel_size)
```

ДОДАТОК Е

Лістинг коду з функціями втрат LSoftmax та CosFace

```
import torch
from numpy import cos, pi, sin
from scipy.special import binom
from torch import nn
from torch.nn import Parameter
from torch.nn import functional as F


class LSoftmaxLinear(nn.Module):
    def __init__(self, in_channels, out_channels, margin):
        super().__init__()
        self.in_channels = in_channels
        self.out_channels = out_channels
        self.margin = margin  # should be in [1; +inf] range

        self.weight = nn.Parameter(torch.FloatTensor(in_channels, out_channels))

        self.divisor = pi / (self.margin + 1e-5)
        self.coeffs = binom(margin, range(0, margin + 1, 2))
        self.cos_exps = range(self.margin, -1, -2)
        self.sin_sq_exps = range(len(self.cos_exps))
        self.signs = [1]
        for _ in range(1, len(self.sin_sq_exps)):
            self.signs.append(self.signs[-1] * -1)

    def reset_parameters(self):
        nn.init.kaiming_normal_(self.weight.data.t())

    def find_k(self, cos):
        acos = cos.acos()
        k = (acos / self.divisor).floor().detach()
        return k

    def forward(self, x, y=None):
        if not self.training:
            assert y is None
            return x.matmul(self.weight)
        assert y is not None
        logit = x.matmul(self.weight)
        batch_size = logit.size(0)
        logit_target = logit[range(batch_size), y]
        weight_target_norm = self.weight[:, y].norm(p=2, dim=0)
        input_norm = x.norm(p=2, dim=1)
        # norm_target_prod: (batch_size,)
        norm_target_prod = weight_target_norm * input_norm
        # cos_target: (batch_size,)
        cos_target = logit_target / (norm_target_prod + 1e-10)
        sin_sq_target = 1 - cos_target**2

        # coeffs, cos_powers, sin_sq_powers, signs: (self.margin // 2 + 1,)
```

```
51          coeffs = torch.tensor(x.data.new(self.coeffs))
52          cos_exps = torch.tensor(x.data.new(self.cos_exps))
53          sin_sq_exps = torch.tensor(x.data.new(self.sin_sq_exps))
54          signs = torch.tensor(x.data.new(self.signs))
55
56          cos_terms = cos_target.unsqueeze(1)**cos_exps.unsqueeze(0)
57          sin_sq_terms = sin_sq_target.unsqueeze(1)**sin_sq_exps.unsqueeze(0)
58
59          cosm_terms = signs.unsqueeze(0) * coeffs.unsqueeze(0) * cos_terms * sin_sq_terms
60          cosm = cosm_terms.sum(1)
61          k = self.find_k(cos_target)
62
63          ls_target = norm_target_prod * (((-1)**k * cosm) - 2 * k)
64          logit[range(batch_size), y] = ls_target
65          return logit
66
67
68  class LargeMarginCosineLoss(nn.Module):
69      # TODO: Move parameters to config
70      def __init__(self, in_channels, out_channels, input_norm=30.0, margin=0.25):
71          super().__init__()
72          self.in_channels = in_channels
73          self.out_channels = out_channels
74          self.input_norm = input_norm  # norm of input feature
75          self.margin = margin  # should be in (0; 1) range
76          self.weight = Parameter(torch.FloatTensor(self.out_channels, self.in_channels))
77          nn.init.xavier_uniform_(self.weight)
78
79      def forward(self, x, y=None):
80          if not self.training:
81              assert y is None
82              return F.linear(F.normalize(x), self.weight)
83          assert y is not None
84          # init cos(theta), phi(theta) was uncovered and included in future transformations
85          cosine = F.linear(F.normalize(x), F.normalize(self.weight))
86          # convert y to one-hot
87          one_hot = cosine.clone().zero_()
88          one_hot.scatter_(1, y.view(-1, 1), self.margin)
89          output = (cosine - one_hot) * self.input_norm
90          return output
91
92
93  class LargeMarginArcLoss(nn.Module):
94      # TODO: Move parameters to config
95      def __init__(self, in_channels, out_channels, input_norm=10.0, margin=0.2, easy_margin=False):
96          super().__init__()
97          self.in_channels = in_channels
98          self.out_channels = out_channels
99          self.input_norm = input_norm  # norm of input feature
100         self.margin = margin  # should be in (0; 1) range
101         self.weight = Parameter(torch.FloatTensor(self.out_channels, self.in_channels))
102         nn.init.xavier_uniform_(self.weight)
103
104         self.easy_margin = easy_margin
```

```
105         self.cos_m = cos(margin)
106         self.sin_m = sin(margin)
107
108     def forward(self, x, y=None):
109         if not self.training:
110             assert y is None
111             return F.linear(F.normalize(x), self.weight)
112         assert y is not None
113         # init cos(theta), sin(theta), phi(theta)
114         cosine = F.linear(F.normalize(x), F.normalize(self.weight))
115         sine = torch.sqrt(1.0 - torch.pow(cosine, 2))
116         phi = cosine * self.cos_m - sine * self.sin_m
117         if not self.easy_margin:
118             phi = torch.where(cosine > -self.cos_m, phi, cosine - self.margin * self.sin_m)
119         else:
120             phi = torch.where(cosine > 0, phi, cosine)
121         one_hot = cosine.clone().zero_()
122         one_hot.scatter_(1, y.view(-1, 1), 1)
123         output = one_hot * phi + (1.0 - one_hot) * cosine
124         return output * self.input_norm
125
126
127 class TripletLoss(nn.Module):
128     def __init__(self, embedding_net):
129         super().__init__()
130         self.embedding_net = embedding_net
131
132     def forward(self, anchor, positive, negative):
133         embedded_anchor = self.embedding_net(anchor)
134         embedded_positive = self.embedding_net(positive)
135         embedded_negative = self.embedding_net(negative)
136         dist_positive = F.cosine_similarity(embedded_anchor, embedded_positive)
137         dist_negative = F.cosine_similarity(embedded_anchor, embedded_negative)
138         return dist_positive, dist_negative
```

ДОДАТОК Ж

Лістинг коду з додатковими шарами нейронних мереж

```
import torch
from torch import nn


class AddCoords(nn.Module):
    def __init__(self, radius_channel=False):
        super().__init__()
        self.radius_channel = radius_channel

    def forward(self, in_tensor):
        # size: batch_size, channels, height, width
        batch_size, _, height, width = in_tensor.size()

        xx_channel = torch.arange(height).type_as(in_tensor).repeat(1, width, 1)
        yy_channel = torch.arange(width).type_as(in_tensor).repeat(1, height, 1).transpose(1, 2)

        xx_channel = xx_channel / (height - 1) * 2 - 1
        yy_channel = yy_channel / (width - 1) * 2 - 1

        xx_channel = xx_channel.repeat(batch_size, 1, 1, 1).transpose(2, 3)
        yy_channel = yy_channel.repeat(batch_size, 1, 1, 1).transpose(2, 3)

        out = torch.cat([in_tensor, xx_channel, yy_channel], dim=1)

        if self.radius_channel:
            rr_channel = torch.sqrt(torch.pow(xx_channel - 0.5, 2) + torch.pow(yy_channel - 0.5, 2))
            out = torch.cat([out, rr_channel], dim=1)

        return out


class CoordConv(nn.Module):
    def __init__(self, in_channels, out_channels, radius_channel=False, **kwargs):
        super().__init__()
        self.addcoords = AddCoords(radius_channel=radius_channel)
        # 2 -> xx, yy ; 3 -> xx, yy, rr
        in_channels += 2 if not radius_channel else 3
        self.conv = nn.Conv2d(in_channels, out_channels, **kwargs)

    def forward(self, x):
        out = self.addcoords(x)
        out = self.conv(out)
        return out


class Flatten(nn.Module):
    def forward(self, x):
        # size: batch_size, channels, height, width
        return x.view(x.size(0), -1)

```

```
51
52  class SequentialWithTarget(nn.Sequential):
53      def forward(self, x, *args):
54          for module in self._modules.values():
55              x = module(x, *args)
56          return x
57
58      def get_scores(self, x):
59          return self.to_numpy(self.predict_proba(x))
60
61      def to_numpy(self, tensor):
62          data = tensor.data.cpu() if tensor.device.type != "cpu" else tensor.data
63          return data.numpy()
64
65      @torch.no_grad()
66      def predict(self, x, device=torch.device("cuda:0")):
67          x = torch.from_numpy(x).permute(0, 3, 1, 2).float().div(255).detach()
68          return self(x.to(device, non_blocking=True, dtype=torch.float32))
69
70      def predict_proba(self, x):
71          return nn.Softmax(1)(self.predict(x))
72
73
74  class SELayer(nn.Module):
75      """
76      SE refers to a Squeeze-and-Excitation Networks (see https://arxiv.org/abs/1709.01507)
77      """
78
79      def __init__(self, channel, reduction=16):
80          super().__init__()
81          self.avg_pool = nn.AdaptiveAvgPool2d(1)
82          self.fully_connected = nn.Sequential(
83              nn.Linear(channel, channel // reduction),
84              nn.ReLU(inplace=True),
85              nn.Linear(channel // reduction, channel),
86              nn.Sigmoid()
87          )
88
89      def forward(self, x):
90          batch_size, channels, *_ = x.size()
91          y = self.avg_pool(x).view(batch_size, channels)
92          y = self.fully_connected(y).view(batch_size, channels, 1, 1)
93          return x * y
```